業務仕様の変化に強いプログラム作成を目指したビジネスロジック・ジェネレータ

Business Logic Generator for Program Generation tolerant of Business Specification Changes

山　本　史　朗

要　約　2012 年に入り，ビジネスライフサイクルの短縮傾向，顧客の IT 投資の冷え込み，システム開発期間の短縮，さらなるコスト圧縮などの外部要因の変化により，システム開発ベンダには，これまで以上の短期開発・低コスト開発が求められている．それに応えるためには，さらなる開発生産性向上への取り組みが必要だが，業務アプリケーションを開発する上で，業務仕様の一致確認作業（単体，機能テスト）や仕様の曖昧さからの手戻り作業が大きな割合を割いている．特に複雑な業務仕様は，仕様確定の漏れや変更を誘発することが多く，開発生産性を向上する上で大きな課題となっている．業務仕様である機能やデータモデルの変更に強い「自動化」手法を用いることで，初期開発時だけでなく，保守フェーズでの作業効率の改善も期待できる．本稿では，日本ユニシスが開発した「自動化」ツールであるビジネスロジック・ジェネレータの概要と機能を紹介し，プロトタイピング事例を踏まえ，業務仕様の変更に強いプログラム作成の試みとして「自動化」手法の有用性について述べる．

Abstract　Since the beginning of 2012, changes in external factors such as the tendency toward the reduced business life cycle, the decrease in the customer's IT investment, and shortened duration of the system development, have forced system development vendors for the further reduced period and costs of development than ever. In order to respond to above requirements, the increased effort for development productivity improvement is essential. However, the business application development spends both the consistency checking of the business specifications (unit testing, functional testing) and the step back working caused by the ambiguities of specification at a great rate. Especially, a complicated business specification often induces the incompleteness and changes in the defined specification, which becomes a major challenge in the development productivity improvement. The use of the "automated" technique tolerant of changes in function and data model in business specifications promises the improvement of working efficiency in not only the early development phase but also the maintenance phase.

This paper introduces the outline and function of the business logic generator as the "automated" tools developed by Nihon Unisys, and then discusses the usefulness of the "automated" technique as an experiment of creating a program tolerant of alteration in business specifications on the basis of a prototyping example.

1. は　じ　め　に

2012 年に入り，ビジネスライフサイクルの短縮傾向，顧客の IT 投資の冷え込み，システム開発期間の短縮，さらなるコスト圧縮などの外部要因の変化により，システム開発ベンダには，これまで以上の短期開発・低コスト開発が求められている．それに応えるためには，さらなる

開発生産性向上への取り組みが必要である．

日本ユニシスは，これまでにも MIDMOST for Java EE（Maia）[*1]/MIDMOST for .NET（Maris）[*2] を用いた業務アプリケーションフレームワークと開発プロセスの標準化などによる開発生産性向上に取り組んできた．しかし，これらは一定の効果は得られているものの，開発フェーズ全体から見た場合の効果は大きくはない．開発生産性向上の別の施策として，プロセスの軽量化（アジャイルプロセス）に取り組む一方で，当社は「自動化」が業務アプリケーション開発の生産性向上にもたらす効果に着目し，業務ロジックの物理設計情報からソースコードを自動生成するツール「ビジネスロジック・ジェネレータ」を開発した．これによりプログラミング工数の削減だけではなく，単体・機能テスト工数も削減することができ，プログラム開発フェーズにおける開発生産性をより確実に向上させることができると考えている．

本稿では，ツールの概要と機能を紹介し，プロトタイピング事例を踏まえ，業務仕様の変更に強いプログラム作成の試みとして「自動化」手法の有用性について述べる．

2. ビジネスロジック・ジェネレータとは

ビジネスロジック・ジェネレータは，物理設計情報から業務ロジックのソースコードを自動生成する機能と，業務ロジックの整合性・妥当性をチェックするテスト機能を有している．本章では，ビジネスロジック・ジェネレータの開発経緯と仕様について述べる．

2.1 ビジネスロジック・ジェネレータの開発経緯

業務アプリケーションを開発する上で，業務仕様との一致確認作業や仕様の曖昧さからの手戻り作業が原因となり，開発生産性を大きく低下させることがある．特に複雑な業務仕様は，判定処理が多く，仕様一致確認のテストパターンが大量になってしまう．また，仕様確定の漏れや変更を誘発することが多く，せっかくテストパターンを作成しても無駄になってしまうこともある．この課題は，初期開発時だけでなく，保守フェーズでの作業効率低下にも当てはまる．

プログラムを作成する以上，業務仕様との一致確認は必要である．通常，一致確認のためには，ディシジョン・テーブルなどを活用してテストケースをバリエーション化し，妥当性を評価する．見方を変えると，このテストバリエーションが設計書の業務仕様として盛り込まれていれば，その設計情報を基にしてプログラムコードを自動生成することも可能となる．つまり「自動化」手法を用いることが解決策の一つとなる．

業務ロジックのプログラムレス化という点では，IBM 社の ILOG JRules やレッドハット社の JBoss Enterprise BRMS のようなビジネスルール管理システム（BRMS）[*3] 製品が既に存在する．これらの製品は，ルールの実行エンジンや開発環境とセットになっており，ソースコードは自動生成せず，他のプログラムモジュールに組み込む場合は，サービス指向アーキテクチャ（SOA）のようなサービス基盤で連携する構造となる．

近年，スクラッチ開発は減少したものの，性能や運用面を配慮して，プログラム部品として利用できる提供形態も今なお必要とされているため，日本ユニシスでは，独自に「自動化」ツールを開発するに至った．

2.2 ビジネスロジック・ジェネレータの特徴と効果

ビジネスロジック・ジェネレータは，業務仕様となる物理設計情報が記述されたExcelファイルから，ソースコードを自動生成するツールである．以下に，特徴と効果を示す．

- プログラミング言語や実データモデルと分離された構造である．
  - 業務仕様の判定ロジック（ルール）に専念できる．
  - 業務仕様の設計データ項目と，実データ項目の整合性を取りやすい．
  - 実データ項目の変更が判定ロジックに影響しにくい．
- 業務仕様をデシジョン・テーブル形式で物理設計書に記述できる．
  - 設計段階で業務仕様を明確にし，曖昧性を排除できる．
  - 業務仕様の設計者と実装者が協力して記述でき，認識齟齬を減らせる．
- 設計段階で，業務仕様の判定ロジックの記述の網羅性などがチェックできる．
  - 判定ロジックの論理的な矛盾や不足条件が検出でき，手戻りを減らせる．
- 設計段階で，物理設計情報として記述された業務仕様をシミュレーションできる．
  - 設計段階で判定ロジックの期待値誤りを検出でき，手戻りを減らせる．
  - テストデータの自動生成により単体テスト相当の作業を削減できる．
- 物理設計情報を入力としてソースコードを自動生成できる．
  - 自動生成によるプログラミングや単体テストの工数を削減できる．
  - 業務仕様とソースコードが同期することにより，保守効率が向上する．

2.3 ビジネスロジック・ジェネレータの構成

ビジネスロジック・ジェネレータは，ビジネスロジック定義ファイル，設計情報ファイル，ソースコード・ジェネレータの三つの要素で構成されている（表1）．

表1　ビジネスロジック・ジェネレータの構成

要素	内容
ビジネスロジック定義ファイル	物理設計情報Excelファイル．判定処理に使用される設計データ項目群や，判定ロジック（ルール）が記述される．
設計情報ファイル	ビジネスロジック定義ファイルに記載された情報をXML形式で保持する．
ソースコード・ジェネレータ※	設計情報ファイルを基に，業務ロジックとなるプログラム・ソースコードを生成する．

※2012年1月現在，Java言語用ジェネレータを提供

ビジネスロジック定義ファイルには，物理設計情報として，判定処理に使用される設計データ項目群（以降，データセット）定義や判定ロジックが記述される．データセットには，判定ロジックで必要となるデータ項目のみ定義し，この項目の値に基づいて条件判定され，合致した場合に記述された項目編集等のアクションが実施される．定義するデータ項目には，対応する実データモデルの項目をマッピング定義することが可能となっている．また，入力補助機能として，関連する外部ファイルの情報や入力可能な情報を入力時に閲覧できるExcelマクロが備わっている．

設計情報ファイルは，ビジネスロジック定義ファイルに記載された情報を構造化し，XML形式で保存したファイルである．保持する情報は，特定のプログラム言語には依存せず，独立

している．

ソースコード・ジェネレータは，設計情報ファイルを基に，業務ロジックとなるプログラム・ソースコードを生成するツールである．複雑な判定およびアクション処理を，物理設計情報に記述されたとおりに実行するソースコードが生成される．

構成要素の関連を図 1 に示す．

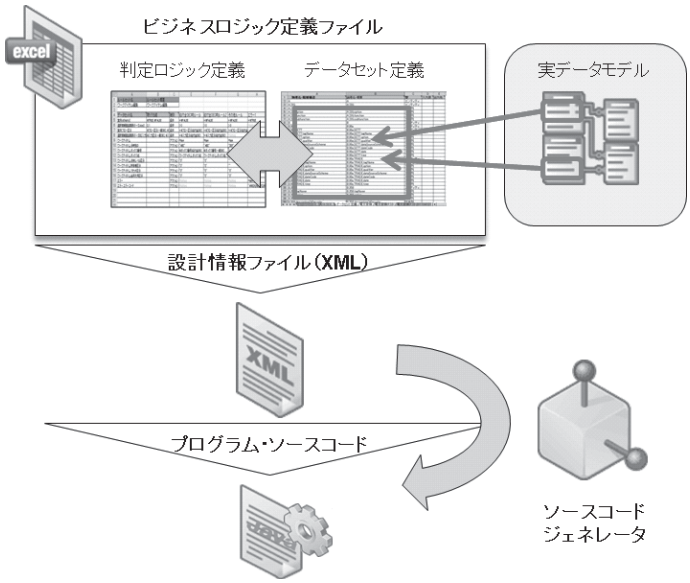


図 1　構成要素の関係

2. 4　業務仕様の記述形式

ビジネスロジック・ジェネレータでは，業務仕様を「ルール（IF-THEN 構造）」，「ルールセット（ルールの集合）」の積み重ねとして，ビジネスロジック定義ファイル（Excel ファイル）に記述する（表 2）（図 2）．

表 2　ビジネスロジック定義の用語説明

用語	内容
ルール	合致する「条件」（IF）と，条件を満たした場合に実行される「アクション」（THEN）から構成される，最小の処理単位．Excel シートの 1 列に相当する．条件やアクションは，あらかじめ定義されている演算子や関数を使用して記述できる．
ルールセット	一つ以上のルール集合であり，1 Excel シートに相当する．条件を満たすルールのアクションを全て実行するか，最初に条件を満たしたルールのアクションのみを実行するか，など判定ポリシーを選択できる．
ビジネスロジック	ルールセットの集合であり，1Excel ブックに相当する．業務仕様（判定ロジック）に必要なルールセットのかたまり．

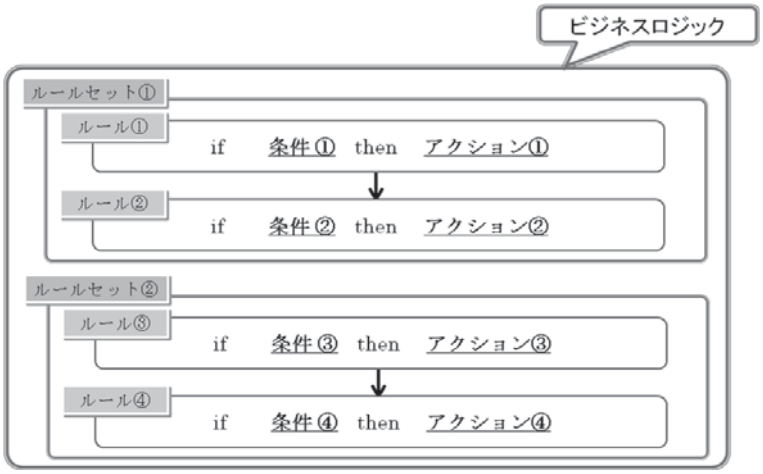


図 2　ルールの定義

ビジネスロジック定義ファイルの記述例を，勤怠管理（労使協定）の基準判定を使って簡単に説明する．判定対象となる項目は，“休日出勤回数（月間）”，“残業時間（月間）”，“残業時間（年間）”となり，それぞれの項目に対する業務仕様は，表 3 に挙げたとおりとする．

表 3　勤怠管理の業務仕様

分類	項目	業務仕様
原則	休日出勤回数（月間）	4回を超えた場合は，判定結果に“休日出勤：原則超過”をする．
限度	残業時間（月間）	80時間を超えた場合は，判定結果に“残業時間（月間）：限度超過”を出力する．
	残業時間（年間）	500時間を超えた場合は，判定結果に“残業時間（年間）：限度超過”を出力する．
上限	残業時間（月間）	120時間を超えた場合は，判定結果に“残業時間（月間）：上限超過”を出力する．
	残業時間（年間）	1000時間を超えた場合は，判定結果に“残業時間（年間）：上限超過”を出力する．

この業務仕様をビジネスロジック定義で記述すると，ルールは以下の五つになる．

ルール①　If（残業時間（月間）＞ 120 時間）　Then（判定結果＝残業時間（月間）：上限超過）
ルール②　If（残業時間（年間）＞ 1000 時間）Then（判定結果＝残業時間（年間）：上限超過）
ルール③　If（残業時間（月間）＞ 80 時間）　Then（判定結果＝残業時間（月間）：限度超過）
ルール④　If（残業時間（年間）＞ 500 時間）　Then（判定結果＝残業時間（年間）：限度超過）
ルール⑤　If（休日出勤回数（月間）＞ 4 回）　Then（判定結果＝休日出勤（月間）：原則超過）

このルール集（ルールセット）を Excel シートに記述すると，表 4 のような記述形式となる．

表 4　ビジネスロジック定義の記述例

データセット項目	タイプ	ルール ①	ルール ②	ルール ③	ルール ④	ルール ⑤
休日出勤回数(月間)	条件					>4
残業時間(月間)	条件	>120		>80		
残業時間(年間)	条件		>1000		>500	
判定結果	アクション	= 残業時間(月間):上限超過	= 残業時間(年間):上限超過	= 残業時間(月間):限度超過	= 残業時間(年間):限度超過	= 休日出勤(月間):原則超過

2. 5　業務仕様の網羅性チェック

ビジネスロジック・ジェネレータには，記述された業務仕様の中で，条件判定ロジックのバリエーション不足や論理矛盾を検出する網羅性チェック機能がある．定義したルールセットのルールについて，不足していると推測されるパターンや，重複したパターンを検出する．網羅性チェック機能により，条件判定ロジックの抜け，漏れを設計段階で検知することができる．

勤怠管理（労使協定）の基準判定の例に，「残業時間（月間）が 60 時間を超え，かつ休日出勤回数（月間）が 4 回を超える場合，「原則超過」であることを出力する」という新たな業務仕様を 6 番目に追加し，ルール⑥をビジネスロジック定義に記述する．この新たなルールセットに対して網羅性チェックを実行すると（表 5），「残業時間（月間）が 60 時間を超える場合」というパターンが不足していると推測され，新たなルールとしてルールセットの末尾に追加される（ルール⑦）．また，ルール⑥よりルール⑤の判定条件が緩く，ルール⑥に到達しない可能性があると推測し，チェック機能コメントとして警告している．

網羅性チェックの実行結果を基に，ルールの評価順序を見直したり，追加されたルールの要・不要を判断した後，必要な場合は「アクション」を記述し，不要な場合はルールを削除（列を削除）する．

表 5　網羅性チェックの実行結果例

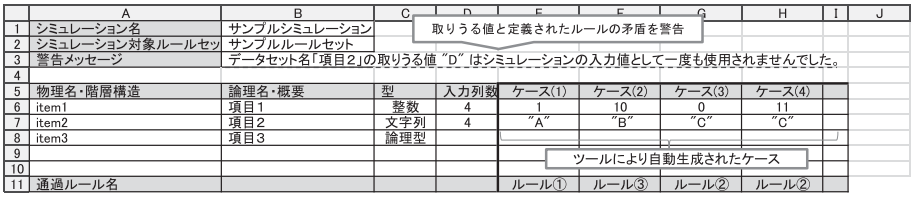
データセット項目	タイプ	ルール ①	ルール ②	ルール ③	ルール ④	ルール ⑤	ルール ⑥	ルール ⑦
休日出勤回数(月間)	条件					>4	>4	
残業時間(月間)	条件	>120		>80			>60	>60
残業時間(年間)	条件		>1000		>500			
判定結果	アクション	= 残業時間(月間):上限超過	= 残業時間(年間):上限超過	= 残業時間(月間):限度超過	= 残業時間(年間):限度超過	= 休日出勤(月間):原則超過	= 残業時間(月間):原則超過	
チェック機能コメント							未到達候補	追加候補

なお，網羅性チェック機能で不足パターンを追加しているが，単純に条件項目の組み合わせを追加するのでは，過度な組み合わせを提示することになる．そのため本チェック機能では，条件指定の傾向を分析し，組み合わせを絞り込む工夫がなされている．

2. 6　処理結果のシミュレーション

ビジネスロジック・ジェネレータには，定義した業務ロジックが想定通りに動作するかを確認するシミュレーション機能がある．動作確認用のデータをビジネスロジック定義ファイルに追記し，シミュレーションを実行すると，結果が Excel ファイルに出力される（図 3）．動作

確認用のデータは，定義したビジネスロジックから自動生成することができ，データ準備の負荷を軽減させることができる（図 4）．シミュレーション機能により，定義した業務ロジックの動作を動的に確認することができ，ロジックの誤りを早期に検知することができる．



	A	B	C	D	E	F	G	H	I	J
1	シミュレーション名	サンプルシミュレーション								
2	シミュレーション対象ルールセッ	サンプルルールセット								
3	警告メッセージ	データセット名「項目2」の取りうる値 "D" はシミュレーションの入力値として一度も使用されませんでした。								
4										
5	物理名・階層構造	論理名・概要	型	入力列数	ケース(1)	ケース(2)	ケース(3)	ケース(4)		
6	item1	項目1	整数	4	1	10	0	11		
7	item2	項目2	文字列	4	"A"	"B"	"C"	"C"		
8	item3	項目3	論理型							
9										
10										
11	通過ルール名				ルール①	ルール③	ルール②	ルール②		

図 3　シミュレーション結果シート例

	A	B	C	D	E
1	ルールセット名	ルールセット概要			
2	サンプルルールセット				
3					
4	データセット名	取りうる値	種別	ルール①	ルール②
5	項目1	0,1 .. 10,11	条件	= [1..10]	!= [1..10]
6	項目2	"A","B","C","D"	条件	= ["A","B"]	!= ["A","B"]
7	項目3		アクション	@TRUE	@FALSE
8					

図 4　シミュレーション機能（取りうる値）例

勤怠管理（労使協定）の基準判定の例で説明すると，月間残業時間が 121 時間の場合，定義したビジネスロジックを実行した結果，判定結果が「上限超過」となる．また，年間残業時間が 120 時間の場合，定義したビジネスロジックを実行した結果，「限度超過」となる（図 5）．

	A	B	C	D	E	F
1	シミュレーション名	サンプルシミュレーション				
2	シミュレーション対象ルールセット	勤怠管理(労使協定)判定				
3	警告メッセージ					
4						
5	物理名・階層構造	論理名・概要	型	入力列数	ケース(1)	ケース(2)
6	holiday_working_number_monthly	休日出勤回数(月間)	整数			
7	overtime_monthly	残業時間(月間)	整数		121	120
8	overtime_year	残業時間(年間)	整数			
9	result_judgment	判定結果	文字列	100	残業時間(月間):上限超過	残業時間(月間):限度超過
10						
11	通過ルール名				ルール①	ルール③

図 5　シミュレーション機能実行例

シミュレーションの実行結果には，ビジネスロジック実行結果とともに，各テストケースで評価されたルール（通過ルール）と，一度も評価されなかったルール（未通過ルール）が出力される．これを基に，定義したビジネスロジックが期待通りの動作をしているかどうかを確認することができる．

2.7　プログラム・ソースコードの生成

ビジネスロジック・ジェネレータには，ビジネスロジック定義ファイルに記述された業務ロジックに相当するソースコードを生成するジェネレータ機能がある．ジェネレータ機能は，定義されたルールセット，ルールを順に評価するようなソースコードを生成する．勤怠管理（労

使協定）の基準判定の例で生成したソースコードを図 6 に示す.

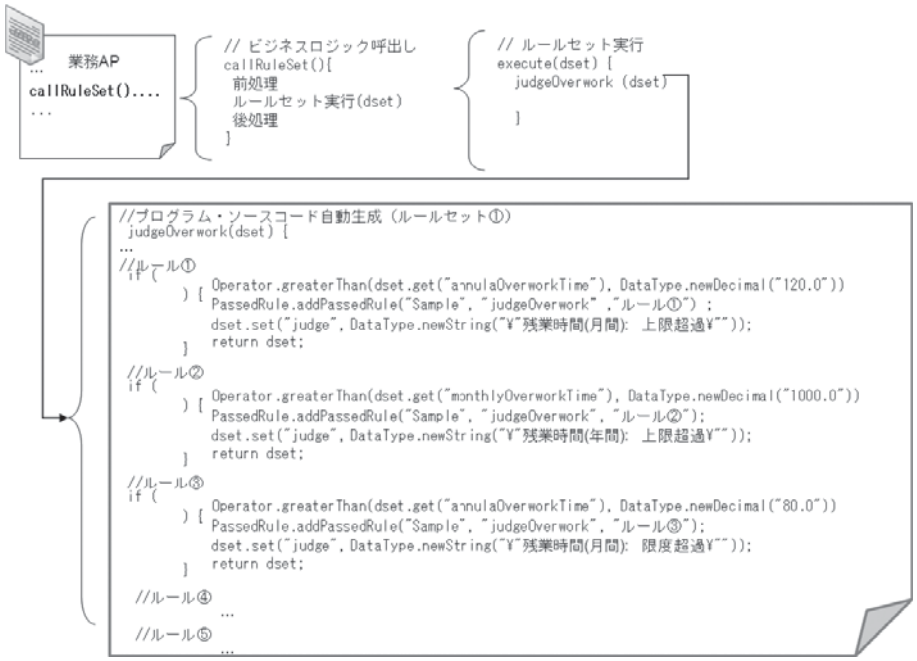


図 6　生成されたソースコード

2. 8　ビジネスロジック・ジェネレータの適用可能領域

ビジネスロジック・ジェネレータを使用しない従来の開発では，業務仕様を文章で物理設計書として記述し，記述された仕様に則ってプログラミングした後，単体テストを行う（図 7）. 文章での仕様記述は，暗黙的な業務知識を前提として説明が省略された，曖昧で“行間の多い”記述となることが多い．そのため，仕様の理解，把握に時間がかかり，実装の抜け漏れが発生

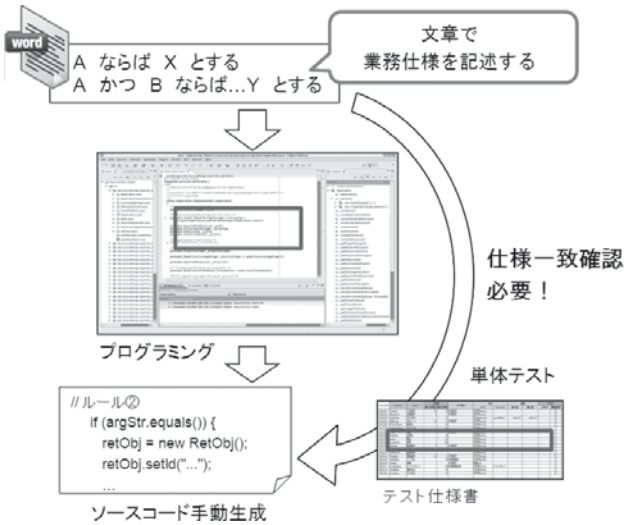


図 7　従来の開発

する可能性がある．また，プログラミングはプログラマのスキルに大きく依存するところがあり，設計書が正しくても正しいソースコードができるとは限らない．単体テストでは，テストケースの洗い出し，仕様との一致性の確認に多くの時間を必要とする．

一方，ビジネスロジック・ジェネレータを使用した開発では，物理設計情報をデシジョン・テーブル形式で記述する．網羅性チェック機能，シミュレーション機能を利用して，設計時に仕様の整合性・妥当性を確認した後，業務ロジックのソースコードを自動生成する（図 8）．デシジョン・テーブル形式での仕様記述は，文章での仕様記述に比べ曖昧性が低く，仕様の理解も容易である．早期に仕様の整合性，妥当性が確認できることにより，手戻りによる影響を少なくすることができる．ソースコードは物理設計情報より完全自動生成されるため，プログラマのスキルに依存することはない．また，単体テストは物理設計時の仕様の整合性・妥当性確認により担保できるため，生成されたソースコードの仕様一致確認は，動作確認程度のテストで済む．

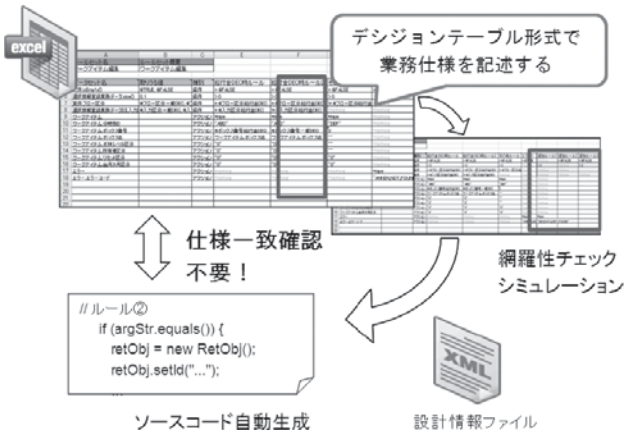


図 8　ビジネスロジック・ジェネレータでの開発

ビジネスロジック・ジェネレータは，業種やアーキテクチャに依存せず業務仕様の全体，もしくは一部に適用可能である．特に，電文変換等のデータマッピング，ワークフローシステムにおける遷移先判定処理，契約判定処理，項目編集/単項目チェックなど，判定処理の多い業務仕様に適している．プログラム実装や単体・機能テストの開発生産性を向上させるだけでなく，設計仕様がプログラムや実データモデルと分離されていることで，実データ項目や判定ロジックの変更に対しても影響を最低限度にとどめることができる．初期開発時だけでなく，保守効率の向上にもつながるため，パッケージソリューションなどの中核業務仕様に適用すれば効果も大きい．

3.　適用効果

ビジネスロジック・ジェネレータ適用時の効果を測定するため，実際の案件に対し，一部機能を対象にプロトタイピングを行った．本章ではプロトタイピングの結果を基に，ビジネスロジック・ジェネレータの適用効果を適用可能性，開発生産性，保守性，品質の視点から述べる．

3.1　適用可能性―電文変換 プロトタイピング事例

現在稼働中の某銀行システムのうち，対外機関との接続に使用される通信電文のフォーマット変換処理（SWIFT MT 形式電文[*4] と業務データの相互変換処理）2 処理に対して，ビジネスロジック・ジェネレータを適用した．現行システムでは，EAI ツールである WebSphere Transformation Extender（以下 WTX）で実現されている機能であり，ビジネスロジック・ジェネレータで WTX と同等の条件判定処理，項目編集処理が実現可能かどうかを評価した（図 9）．

図 9　プロトタイピング事例　電文変換処理概要

プロトタイピングの結果，表 6 に示すとおり，条件判定処理，項目編集処理ともに，ビジネスロジック・ジェネレータで実現できなかった処理はなかった．対象機能では，条件判定，項目編集ともに種類は多くないが，電文変換のツールとして適用可能であり，WTX の代替ツールとして検討の価値があるものと評価できる．

表 6　電文変換での条件判定処理数と項目編集処理数

処理内容		処理数
条件判定処理		105
内訳	値の一致・不一致判定	19
	値の有無判定	53
	項目長の大小判定	27
	配列項目に対する値の一致・不一致要素の有無判定	5
	配列項目長の大小判定	1
	変換できなかった処理	0
項目編集処理		306
内訳	固定値	115
	書式整形済み日時	27
	入力項目と同値	160
	入力項目の演算・編集結果	14
	変換できなかった処理	0

3.2　開発生産性―ワークフローシステムプロトタイピング事例

2011 年に本番を迎えた，ワークフローシステムを使った保険契約処理システムの機能の一部にビジネスロジック・ジェネレータを適用し，実際の開発時の作業工数と比較した．対象と

した処理はワークフローの遷移先を判定する処理であり，対象機能の開発作業工数中の約 1/4 を占める（図 10）.

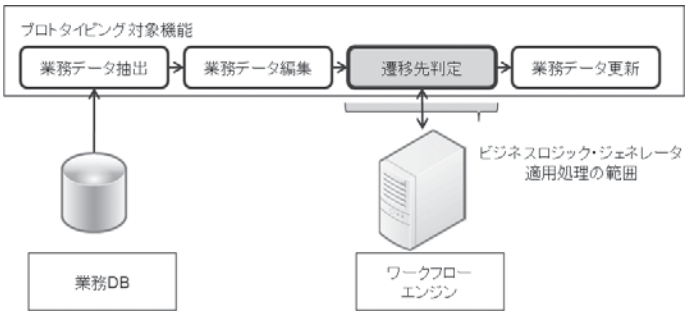


図 10　プロトタイピング事例　ワークフローシステム処理概要

プロトタイピングの結果，表 7 に示すとおり，物理設計書作成における工数は増加したものの，全体として約 18%の工数が削減された．ビジネスロジック・ジェネレータでは，物理設計情報の作成，網羅性チェック，シミュレーションを物理設計時に行うため，物理設計の作業工数は従来よりも増加する．しかし，プログラミング，単体テストでそれ以上の工数を削減できており，全体的に開発生産性は向上している．ビジネスロジック・ジェネレータを適用する範囲を拡大することにより，さらなる開発生産性向上が見込める．

表 7　作業工数比較

作業分類	実開発時（人日）	ビジネスロジック・ジェネレータ適用時（人日）（カッコ内は増減比）
物理設計書作成	2.0	2.5　(25%)
プログラミング	15.0	11.5　(△24%)
単体テスト仕様書作成	5.0	4.0　(△20%)
単体テスト実施	4.0	3.25　(△18%)
合計	26.0	21.25　(△18%)

3.3　保守性

ビジネスロジック・ジェネレータでは，物理設計情報とソースコードが常に同期しているため，改修時のコストの削減や，設計書とソースコードの不整合による事故の防止に繋がる．また，修正に対する影響範囲が，物理設計情報だけで確認できることも重要である．自動生成されたソースコードは可読性が高いため，仮にビジネスロジック・ジェネレータが利用できなくなったとしても，ソースコード自体を保守していくことも可能である．

3.4　品質

整合性チェック・網羅性チェック・シミュレーションなどの機能により，ロジックの抜け・漏れ，矛盾が少なくなり，実質的に単体テスト済みと見なすことのできる，高品質なソースコードが生成される．

物理設計情報の作成に論理名称を使用できることから，物理設計情報のレビューもしやすくなり，不具合の早期検出につながる．特に，大規模なウォーターフォール開発では，下流工程

でのバグのすり抜け率が下がり，大きな手戻りがなくなり，システム全体の品質が上がることが期待できる．

4. お わ り に

開発生産性を向上させる施策として，プロセスの軽量化（アジャイルプロセス）も進んでいる．開発規模やシステム要件の難易度にもよるが，開発期間が短ければ短いほど，業務仕様の変更は手戻り作業となり致命傷になり得る．変更をある程度許容し，保守フェーズにおいても作業効率を改善する施策として，「ビジネスロジック・ジェネレータ」による「自動化」手法について説明した．データセットにより，実データモデルから分離し，ルール定義により，判定ロジックをプログラマから業務設計者が中心となって実施でき，変更を許容しつつ，業務仕様を明確に定義・実装できる手法である．

適用事例数は少ないものの，プロトタイピング事例により複雑な業務仕様についてもルール定義が可能なことが証明された．ビジネスロジック・ジェネレータを適用することにより，開発生産性を向上させるだけでなく，品質，保守性の向上にも寄与できると評価している．

今後の課題と対策を以下に挙げる．

- 設計・実装・単体テストの方法が変わることへの不安感の払拭
  - ➢ 開発手順のガイドライン策定
- まだ適用案件が少なく，決まった業種に留まっている
  - ➢ プロトタイピング，パイロット開発の推進
- 業務共通機能等のルール・テンプレート化と展開
  - ➢ ナレッジとしてストック，横展開の推進

これらの課題を解決しつつ，開発生産性のさらなる向上を目指していきたい．

最後に，本論文の執筆にあたり，多大なご教示・ご協力を賜りました．深く感謝し，厚く御礼を申し上げます．

＊1　日本ユニシスの Java アプリケーション開発標準
＊2　日本ユニシスの .NET アプリケーション開発標準
＊3　業務上の規則や判断基準，経験的な対処パターンをビジネスルールとして定義・登録し，その組み合わせから複雑な業務判断を自動的に行うコンピュータ・システムのこと
＊4　SWIFT（国際銀行間通信協会）により定められている，金融機関間での通信メッセージの形式

執筆者紹介　山　本　史　朗（Fumiaki Yamamoto）
1991 年日本ユニシス(株)入社．Java 黎明期よりアプリケーションアーキテクトとしてシステム開発支援に従事．現在，システム統括部 AP 基盤技術室長として日本ユニシスの開発標準「MIDMOST for Java EE Maia」「MIDMOST for .NET Maris」の開発，適用，教育を担当．