Vol.2013-SE-179 No.19
2013/3/11

オープンソースミドルウェアへの移行のための移植開発支援ツールの評価

毛受崇[†]　山本恭弘[†]　岡野真一[†]

既存システムで使用している商用ミドルウェアをオープンソースに移行することによってライセンス費用やサポート費用の削減が見込める．しかし，商用ミドルウェア上で動作していたアプリケーションやデータベースをオープンソースのミドルウェア上へ移植するための開発費用が，移行により削減可能な費用を上回る恐れがある．そこで，移植開発費用の削減を目的として，移植開発支援ツールの有効性を評価した．数種類のツールを選定して既存システムに適用し，各ツールの能力を網羅性，正確性，および自動化度の観点から定量的および定性的に評価した．また，ツールによる稼働量削減効果を試算した．

Evaluating Porting Support Tools for Migration to Open Source Middleware

TAKASHI MENJO[†]　YASUHIRO YAMAMOTO[†]　SHIN'ICHI OKANO[†]

Migration from proprietary middleware to open source one can reduce license and support fees of existing systems. However, the cost of porting applications and databases running on the proprietary middleware to the open source middleware can exceed the reduced fees. For cutting porting cost, we evaluated the effectiveness of porting support tools. We chose several tools, applied them to our existing systems, and evaluated their abilities from the aspects of coverage, accuracy, and automaticity, quantitatively and qualitatively. We also estimated how effective they could be for cutting manpower.

1. はじめに

情報システムの開発には巨額の投資が必要である．価格競争力のあるサービスを将来にわたって提供し続けるためには，システムの TCO の削減が課題である．我々はこれらの課題の解決のために，オープンソースの積極的な活用に取り組んでいる．例えば，新規システムの開発において商用ミドルウェアの代わりにオープンソースのミドルウェアを採用することにより，情報システムのライセンス費用やサポート費用の削減を図っている．

新規システムだけでなく，既存のシステムについても，システムの EoL などの契機で商用ミドルウェアをオープンソースに移行することにより，ライセンス費用やサポート費用の削減が見込める．一方，移行に伴って，商用ミドルウェア上で動作していたアプリケーションやデータベースをオープンソースのミドルウェアに移植する必要がある．この移植のための開発費用が単純更改と比較して増大すると，オープンソースに移行しても，トータルで見た費用が削減できない恐れがある．

我々は移植開発費用の削減を目的として，移植開発支援ツール（以下，単に「ツール」とも記す）の有効性を評価した．数種類のツールを選定して既存システムに適用し，各ツールの能力を定量的および定性的に評価した．また，ツールによる移植開発における稼働削減効果を試算した．

本稿の構成は以下の通りである．2 節では移行対象のミドルウェアについて述べ，3 節ではそれらミドルウェアで利用可能な移植開発支援ツールについて述べる．4 節ではツールの評価手法について述べ，5 節ではツールの評価実験とその結果について述べる．6 節では評価結果に基づいた考察を行い，今後の課題について述べる．最後に 7 節でまとめと今後の取り組みを述べる．

2. 移行対象のミドルウェア

ミドルウェアには高可用クラスタ，データベースサーバ，アプリケーションサーバ，システム運用管理など，様々な種類がある．その中でも，アプリケーションサーバとデータベースサーバを対象とする．これらの移行時にはアプリケーションのソースコードおよび構成ファイル，ならびにデータベース・オブジェクトやそれらにアクセスするための SQL コードの修正が必要となり，他の種類のミドルウェアと比較して開発費が増大しやすいからである．

移行元の商用アプリケーションサーバおよびデータベースサーバとしては Sun Java System Application Server のバージョン 8.x 系および Oracle データベースのバージョン 9i 以降を対象とする．また，移行先としては GlassFish 3.1.2.2 および PostgreSQL 9.1.6 を対象とする．

3. 移植開発支援ツール

2 節にて述べたミドルウェアの移行を行う際に修正が必要となる項目，およびそのために利用可能なツールの一覧

† 西日本電信電話株式会社
Nippon Telegraph and Telephone West Corporation

表 1　評価対象の移植開発支援ツール

ツール名	構成ファイル	データベース・オブジェクト	データベース中のデータ	SQLコード
asupgrade	○	－	－	－
Ora2Pg	－	○	○	－
db_syntax_diff	－	－	－	○

（「○」：対応，「－」：未対応）

表 2　Ora2Pg が対応しているデータベース・オブジェクト

データベース・オブジェクト	補足
表（TABLE） ・列（型，NULL 性，DEFAULT 式） ・主キー制約（PRIMARY KEY） ・一意キー制約（UNIQUE） ・外部キー制約（FOREIGN KEY） ・その他の制約（CHECK）	-
索引（INDEX）	-
ビュー（VIEW）	-
実体化ビュー（MATERIALIZED VIEW）	VIEW，TABLE，および FUNCTION の組み合わせに書き換え
権限（ROLE）	-
順序（SEQUENCE）	-
表領域（TABLESPACE）	-
トリガー（TRIGGER）	-
関数（FUNCTION)	-
手続き（PROCEDURE）	FUNCTION に書き換え
パッケージ（PACKAGE）	FUNCTION に書き換え
パーティション（PARTITION）	継承付きの TABLE，TRIGGER，および FUNCTION の組み合わせに書き換え
ユーザ定義型（TYPE）	-
データ	INSERT 文または COPY 文として出力

を表 1 に示す．これらのツールが本稿の評価対象である．以下，各ツールの詳細を述べるとともに，本稿では評価を行わないツールについても触れる．

3.1 asupgrade

asupgrade [1] は GlassFish に付属するオープンソースのツールであり，Sun Java System Application Server または旧バージョンの GlassFish 上で動作するアプリケーションの構成ファイルを新しいバージョンの GlassFish 上で動作するように修正するツールである．使用したバージョンは GlassFish 2.1.1 および同 3.1.2.2 付属のものである（以下，それぞれ「v2 ツール」「v3 ツール」と記す）．前者は Sun Java System Application Server から GlassFish 2.1.1 への修正に，後者は GlassFish 2.1.1 から同 3.1.2.2 への修正に対応している．両者を組み合わせることによって，Sun Java System Application Server から GlassFish 3.1.2.2 へアプリケーションの構成ファイルを修正できる．

3.2 Ora2Pg

Ora2Pg [2] は Gilles Darold が開発しているオープンソースのツールであり，稼働中の Oracle データベースからデータベース・オブジェクトの定義およびデータ（すなわち表内の行）を PostgreSQL 向けの SQL コードとして修正し出力するツールである．使用したバージョンは 9.2 であり，対応しているデータベース・オブジェクトは表 2 の通りである．PostgreSQL は Oracle データベースが持つ実体化ビュー，手続き，パッケージ，およびパーティション機能を持たないため，Ora2Pg はこれらを同表の補足に示す PostgreSQL のデータベース・オブジェクトに書き換える．これにより，Oracle データベースの各機能と同等の機能を PostgreSQL 上で実現する．

3.3 db_syntax_diff

db_syntax_diff [3] は NTT が開発しているオープンソースのツールであり，Oracle データベース向けに記述された SQL コードを入力として，それを PostgreSQL へ移植するために，コード中の修正する必要がある箇所を「非互換 ID」，修正方法，およびコード上の場所（行・列番号）の三つ組として出力する．本稿ではこの三つ組を「非互換項目」と呼ぶ．非互換 ID と修正方法は一対一に対応しており，同じ非互換 ID を持つ非互換項目の修正方法は同等である．db_syntax_diff は Ora2Pg が対応していない SELECT 文や UPDATE 文などのデータ操作を行う SQL にも対応している．また，Java などの他のプログラミング言語の中に文字列リテラルとして埋め込まれた SQL コードに対しても適用できる．ただし，実際の修正そのものは人手によって行う必要がある．使用したバージョンは GitHub にて 2012 年 11 月 22 日にコミットされたものである．

3.4 本稿では評価を行わないツール

asupgrade の類似ツールとして asmigrate [4] が存在する．これは Sun Java System Application Server 7 以前をサポートするが，今回の移行元である 8.x 系はサポート外である．そのため，本稿では評価の対象外とする．

Ora2Pg や db_syntax_diff の類似ツールとして，Postgres Plus Advanced Server Migration Studio [5] など，いくつかの有償ツールが存在する．これらは有償のため導入の敷居が高く，またオープンソースでないため本稿の評価結果をフィードバックすることが難しい．そのため，本稿では評価の対象外とする．

4. ツールの評価方法

ツールは下記三つの観点から定量的および定性的に評価する．

網羅性
移植時に修正する必要がある箇所を，ツールがどれだけ漏れなく抽出できるか

正確性
ツールが行う指摘および変換がどれだけ正しいか

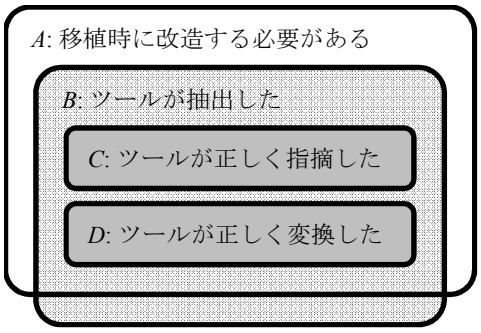

図 1　修正項目集合間の関係

自動化度

移植時に修正する必要がある箇所を，ツールがどれだけ漏れなくかつ正しく変換できるか

上記三つの観点を定量的に定義する．移植時に修正する必要がある箇所を要素とする集合を *A*，ツールが抽出した箇所を要素とする集合を *B*，ツールが正しく指摘または変換した箇所を要素とする集合をそれぞれ *C*，*D* とする．本稿ではこれらの集合を「修正項目集合」と呼ぶ．4 つの修正項目集合は図 1 に示すオイラー図のような関係になる．これらの集合を用いて，前述した三つの観点を次式により定義する. なお, 各集合の要素はツールごとに別途定める．

$$\text{網羅性} = \frac{|A \cap B|}{|A|}$$

$$\text{正確性} = \frac{|C| + |D|}{|B|}$$

$$\text{自動化度} = \frac{|D|}{|A \cup B|} = \frac{|D|}{|A| + |B| + |A \cap B|}$$

加えて，ツールによる移植開発における稼働削減効果を試算する. この試算結果を実際の開発の条件（開発ベンダ，開発に携わる技術者の技術水準）にあてはめることによって，開発費の削減効果を見積もることが可能になると考える．

4.1 asupgrade の評価方法

asupgrade の評価では，構成ファイルに対する編集作業を修正項目集合の要素とする．asupgrade が入出力とするアプリケーションの構成ファイルはいずれも XML 形式である．したがって，ツールが行うことは，ある木構造データから別の木構造データへの一連の編集作業と見なせる．本稿では編集作業を以下のように定める．

- 要素の追加，移動，または削除
- 要素のテキスト値の追加，変更，または削除
- 属性の追加，削除，または値の変更

評価では，ツールを適用する前後の構成ファイルを比較し, その差分を編集作業の集合と見なす（これが *B* である）．そして，これらの作業が適切に行われたものであるか（*D* に含まれる）そうでないか（$B \cap \overline{C} \cap \overline{D}$ に含まれる）を判断する．また，アプリケーションを新しい環境で稼働させるために，ツール適用後の構成ファイルに人手によって行った追加系の編集作業（要素の追加，要素のテキスト値の追加，および属性の追加）は $A \cap \overline{B}$ に含める．これを v2 ツールと v3 ツールの両方について行う．理想的には，ツールを適用する構成ファイルは実際にアプリケーションを稼働させているものそのままが望ましい．しかし，現実には，ツールの制約上，ツール適用前にもいくつかの編集作業が必要となる．これらは $A \cap \overline{B}$ に含める．

XML の仕様を尊重して，属性の順序の差は無視する．また，構成ファイルの DTD を尊重して，順序が重要でないと定義されている要素については順序の差を無視し，要素の移動は行われなかったものと見なす．ツール適用前後で同じ名前の要素が複数存在する場合には，その要素自身と親要素の ID に相当する属性の値が等しいものを対応する要素と見なす.

asupgrade の自動化度は正確性と等しい．これは，当該ツールは機械的な書き換えのみを行い，修正する必要がある箇所の指摘は行わないツール（すなわち，*C* が空集合）であるためである．

4.2 Ora2Pg の評価方法

Ora2Pg の評価では, 同ツールが対応しているデータベース・オブジェクトごとに表 3 のような正誤判断基準を定める．これらに加えて，全てのデータベース・オブジェクトについて「名前が等価である」という基準を設ける．そして，個々のデータベース・オブジェクトを修正項目集合として評価を行う.

Ora2Pg の自動化度は正確性と等しい．これは，当該ツールは機械的な書き換えのみを行い，修正する必要がある箇所の指摘は行わないツール（すなわち，*D* が空集合）であるためである．

4.3 db_syntax_diff の評価方法

db_syntax_diff の評価では，当該ツールが出力する非互換項目相当のものを修正項目集合の要素とする．移植時に修正する必要があるがツールが抽出できなかったもの（すなわち $A \cap \overline{B}$ ）については，修正の類型ごとに非互換 ID を新たに割り当てる．

db_syntax_diff の自動化度はゼロである．これは，当該ツールは修正する必要がある箇所の指摘のみを行い，機械的な書き換えは行わないツール（すなわち，*D* が空集合）であるためである．

5. 実験および評価

移植開発ツールの評価を行うための実験を行った．弊社が顧客に対してネットワークサービスを提供するための 2 システム（以下，それぞれ「システム甲」「システム乙」と

表 3　Ora2Pg の正誤判断基準

データベース・オブジェクト	正誤判断基準
表	● 列の順序が等しい ● 各種制約（主キー，一意キー，外部キー，その他）が等しい
列	● NOT NULL 制約が等価である ● DEFAULT 式が等価である ● Oracle データベースで格納できた値が PostgreSQL でも格納できるような型の書き換えを行っている
索引	● 列の順序が等しい ● 索引の種類（B 木やハッシュなど）が等しい
順序	● 増分値，最大値，最小値，CYCLE 性，およびキャッシュサイズが等価である
表領域	● 表領域を格納するディレクトリパスが適切に設定されている ● 表および索引との紐付けが等価である
関数および手続き	● 本体部分が等価である
ユーザ定義型	● 本体部分が等価である
ビュー	● 本体部分が等価である
実体化ビュー	● 本体部分が等価である ● 更新方法（フルまたは増分）および更新契機が等価である

表 4　システムが使用するミドルウェアのバージョン

	アプリケーションサーバ	データベースサーバ
システム甲	Sun Java System Application Server 8.1	Oracle 9i
システム乙	未使用	Oracle 10g

記す）をサンプルとして，3 節で述べたツール群を適用した．サンプルシステムが使用するミドルウェアのバージョンは表 4 の通りである．

5.1 asupgrade の評価

システム甲に対して asupgrade を適用した．定量的な評価結果を表 5 に示す．ツールの特性より，$|C|$ はゼロであり，また正確性は自動化度と等しいため，表では省略する．

以下，定性的な評価について述べる．

(1) ツール未対応の項目

v2 ツールが書き換えに対応していない項目が 9 項目存在した．これらは v2 ツール適用前に手作業により書き換えた．

(2) 環境に依存する項目

v2 ツール適用後の構成ファイルには，アプリケーションを配置するディレクトリのルート位置や Java ランタイムのクラスパス設定など，アプリケーションサーバの動作環境に依存する項目が 34 項目存在した．これらは v2 ツール適用後に手作業によって書き換えた．

表 5　システム甲に対する asupgrade の評価結果

	$\|A\|$	$\|B\|$	$\|A \cap B\|$	$\|D\|$	網羅性	自動化度
v2 ツール	307	298	298	264	0.97	0.89
v3 ツール	1110	1110	1110	1110	1	1

表 6　システム甲に対する Ora2Pg の評価結果

	$\|A\|$	$\|B\|$	$\|A \cap B\|$	$\|D\|$	網羅性	自動化度
表	83	83	83	83	1	1
列	605	605	605	601	1	0.99
索引	12	12	12	12	1	1
順序	2	2	2	2	1	1
表領域	1	1	1	0	1	0
手続き	3	3	3	0	1	0
関数	1	1	1	0	1	0
ユーザ定義型	2	3	2	0	1	0

表 7　システム乙に対する Ora2Pg の評価結果

	$\|A\|$	$\|B\|$	$\|A \cap B\|$	$\|D\|$	網羅性	自動化度
表	19	29	19	19	1	0.66
列	112	112	112	109	1	0.97
ビュー	15	15	15	11	1	0.73
実体化ビュー	5	5	0	0	1	0
順序	10	10	10	10	1	1
表領域	2	2	2	0	1	0
手続き	12	12	12	0	1	0
関数	5	5	5	0	1	0
ユーザ定義型	0	1	0	0	N/A	0

5.2 Ora2Pg の評価

システム甲およびシステム乙に対して Ora2Pg を適用した．定量的な評価結果を表 6 および表 7 に示す．表では各システムが使用していないデータベース・オブジェクトを省略している．また，ツールの特性より，$|C|$ はゼロであり，また正確性は自動化度と等しいため，表では省略する．

以下，定性的な評価について述べる．

(1) 表

システム甲の全ての表について，表名，列の順序，および各種制約は正しく修正された．

システム乙では，移植時に修正する必要がある 19 の表に加えて，名前が「rupd$_」または「mlog$_」で始まる表がそれぞれ 5 つ，合計 10 出力された（これらが $\overline{A} \cap B$ である）．これらの表は CREATE MATERIALIZED VIEW LOG 文によって Oracle データベースが自動的に作成し，内部的に使用する表であったため，移植は不要であった．なお，表

7 に示す「列」の評価結果では，*B* からこれらの表に含まれる列を除いている．

(2) **列**

列の型は表 8 のように書き換えられた．NVARCHAR2(n) 型から varchar(n) 型への書き換え，および TIMESTAMP 型から timestamp 型への書き換えは，等価な型への書き換えのため，適切と判断する．それ以外の型書き換えについては以下のように判断する．

表 8　Ora2Pg による型書き換え

Oracle データベース		PostgreSQL		システム甲における列数	システム乙における列数	適・不適判断
CHAR(n)	固定長文字列型（n バイトまたは n 文字）	char(n)	固定長文字列型（n 文字）	125	27	適
VARCHAR2(n)	可変長文字列型（最大 n バイトまたは n 文字）	varchar(n)	可変長文字列型（最大 n 文字）	378	56	適
NVARCHAR2(n)	可変長文字列型（最大 n 文字）	varchar(n)	可変長文字列型（最大 n 文字）	0	6	適
CLOB	文字ラージ・オブジェクト型	text	可変長文字列型（長さ上限なし）	12	0	適
DATE	日付型（秒単位，タイムゾーンなし）	timestamp	日付型（マイクロ秒単位，タイムゾーンなし）	22	16	適
TIMESTAMP	日付型（マイクロ秒単位，タイムゾーンなし）	timestamp	日付型（マイクロ秒単位，タイムゾーンなし）	0	3	適
NUMBER(n) [1≦n≦4]	整数型（n 桁）	smallint	2 バイト符号付き整数型	55	3	適
NUMBER(n) [5≦n≦6]	整数型（n 桁）	integer	4 バイト符号付き整数型	9	0	適
NUMBER(n) [n=10]	整数型（n 桁）	bigint	8 バイト符号付き整数型	0	1	適
NUMBER	浮動小数点型（精度 38 桁）	bigint	8 バイト符号付き整数型	4	0	不適

- CHAR(n) 型および VARCHAR2(n) 型から char(n) 型および varchar(n) 型への書き換えは適切と評価する．Oracle データベースでは n はコンテキストに応じてバイト数もしくは文字数を表すが，PostgreSQL では常に文字数である．したがって，Oracle データベースで格納できる文字列は全て PostgreSQL でも格納できる．
- CLOB 型から text 型への書き換えは適切と評価する．PostgreSQL には CLOB 型が存在せず，意味的に最も近い型は text 型のためである．
- DATE 型から timestamp 型への書き換えは適切と評価する．PostgreSQL において秒単位と同等またはより細かい解像度を持つ日付型は timestamp 型のみのためである．（PostgreSQL の date 型は日単位の解像度しか持たない．）
- NUMBER(n) 型から smallint 型，integer 型，または bigint 型への変換は適切と評価する．いずれの n の場合でも，Oracle データベースで格納できる数値は全て PostgreSQL でも格納できるためである．
- NUMBER 型から bigint 型への書き換えは不適切と評価する．この書き換えによって小数点数を格納できなくなるためである．小数点数を格納できる numeric 型への書き換えが適切であると考える．

システム甲では型が不適切に書き換えられた列が 4 列存在した．いずれも NUMBER 型から bigint 型への変換であった．

システム乙では型が不適切に書き換えられた列は存在しなかった．しかし，NOT NULL 制約と DEFAULT 式の両方を持つ列から NOT NULL 制約が削除されて書き換えられた列が 3 列存在した．さらに，それらのうちの 1 列は DEFAULT 式中に含まれていた Oracle データベース特有のキーワードが書き換えられなかった．

(3) **索引**

システム甲の全ての索引は正しく修正された．なお，システム乙は索引を使用しないシステムであった．

(4) **順序**

Oracle データベースと PostgreSQL では順序の最大値に関する仕様が異なる．Oracle データベースでは $10^{28}-1$，PostgreSQL では $2^{63}-1$ であり，後者は前者より小さい．順序の定義時に最大値を明示しなかった場合にもこれらの値が用いられる．

Ora2Pg はいくつかの順序の定義について，最大値に関する箇所を以下のように書き換えた．ここで，書き換え前の値は $10^{28}-1$ に，書き換え後の値は $2^{63}-1$ に等しい．

書き換え前

```
MAXVALUE 9999999999999999999999999999
```

書き換え後

```
NO MAXVALUE
```

この書き換えはデータベース間の仕様差分上，適切と判断する．順序から取得した値を主キー値として用いる場合など，一般的な使用方法においては問題ないと考える[a].

Ora2Pg はまた，順序の定義から ORDER 句を削除した．PostgreSQL では ORDER 句は使用不可なので，この書き換えも適切と判断する．

(5) **表領域**

いずれのシステムについても，Ora2Pg は移植すべき表領域を表および索引と適切に紐付けて書き換えた．ただし，表領域のディレクトリパスは書き換えられなかったため，PostgreSQL の動作環境に合わせて人手により修正する必要があった．

(6) **手続きおよび関数**

システム甲およびシステム乙の全ての関数および変数について，Ora2Pg は関数の宣言を PostgreSQL 向けに適切

a) 仮に，1 秒間に 10 億個の値を順序から取得した場合，1 から開始して $2^{63}-1$ に到達するまでに 290 年以上かかる．

に修正した．しかし，関数の引数および戻り値の型，ならびに関数内で使用されている変数の型については，表 8 に示したような適切な書き換えと不適切な書き換えが混在していた．また，関数の実際の処理部分については，前述の Oracle データベースに特有のキーワード，関数，および文法の一部を書き換えるに留まった．これらを PostgreSQL 上で動作させるためには人手による修正が必要であった．

(7) **ユーザ定義型**

システム甲は 2 つのユーザ定義型を使用していた．これらは型の名前のみが異なり，型の内容はどちらも同じ配列型（VARRAY）であった．これらの型は Ora2Pg によっていずれも以下のように書き換えられた．

書き換え前

```
CREATE TYPE TYPE_NAME IS
VARRAY(100) OF VARCHAR2(30);
```

書き換え後

```
CREATE TYPE type_name AS
(TYPE_NAME VARCHAR2(30)[100]);
```

この書き換えには二点の問題がある．まず，PostgreSQL に VARCHAR2 型は存在しないため，VARCHAR 型に修正する必要があった．さらに，上記は PostgreSQL の複合型として書き換えられているため，手続きや関数などの中でこの型の列や変数を使用する箇所に属性名 *TYPE_NAME* を書き加える必要があった．

いずれのシステムについても，移植が不要であるにも関わらず Ora2Pg が出力した(すなわち $\overline{A} \cap B$ なる)型として，REPCAT$_OBJECT_NULL_VECTOR 型が存在した．これは Oracle データベースがレプリケーション管理のために用いる型のひとつである．

(8) **ビュー**

Ora2Pg はシステム乙のビューの定義を部分的に書き換えて出力した．実験により確認できた書き換えで，不適切と判断したものを以下に示す．

- DECODE 関数が CASE 式に書き換えられた．DECODE 関数の第一引数または第二引数に NULL が渡される場合，書き換え前後で評価結果が異なる可能性がある．
- FROM 句中の副問い合わせの SELECT 文が PERFORM 文に書き換えられた．SELECT 文に戻す必要がある．

また，Ora2Pg は KEEP 句および外部結合演算子「(+)」について書き換えを行わなかった．これらはいずれも PostgreSQL では使えないキーワードを含むため，人手によって書き換える必要があった．

上記の項目のいずれかを含むビューは 4 つ存在した．

(9) **実体化ビュー**

Ora2Pg は実体化ビューを PostgreSQL の表として実現するように Oracle データベースの実体化ビュー定義を書き換えた．具体的には，実体化ビュー定義を通常のビュー定義として書き換えるとともに，実体化ビューに相当する表を作成，削除，および更新する関数，ならびに実体化ビューに相当する表の名前を管理する表を追加した．

Ora2Pg によって PostgreSQL 向けに書き換えられたシステム乙の実体化ビューはフル更新（全ての行を削除して再読み込み）を行うものであった．また，更新のタイミングはユーザが改めて定める必要があった．システム乙において使用されていた 5 つの実体化ビューはいずれも一定の時間間隔で増分更新を行う実体化ビューであったため，Ora2Pg による上記の実現方法は書き換えとして適切ではないと判断する．

5.3 db_syntax_diff の評価

システム甲に対して db_syntax_diff を適用した．定量的な評価結果を表 9 に示す．ツールの特性より，$|D|$ および自動化度はゼロであるため，表では省略している．

表 9　システム甲に対する db_syntax_diff の評価結果

$\|A\|$	$\|B\|$	$\|A \cap B\|$	$\|C\|$	網羅性	正確性
102	313	92	92	0.90	0.29

移植時に修正が必要だが db_syntax_diff が指摘できなかった箇所（すなわち $A \cap \overline{B}$）は 10 箇所であった．これらは以下に示す 5 種類に類型できた．

(1) **副問い合わせを用いた複数列の更新**

Oracle データベースでは副問い合わせを用いて複数列を更新する UPDATE 文が利用可能だが, PostgreSQL では利用不可能である．このような文は副問い合わせを用いない形式の文に修正する必要がある．

(2) **長さゼロの文字列と NULL の区別**

Oracle データベースは長さゼロの文字列を NULL として扱うが，PostgreSQL は両者を区別する．したがって，長さゼロの文字列定数は NULL に修正する必要がある．

(3) **日付と数値の加減算**

5.2 節の(2)にて述べたように，Oracle データベースの DATE 型は PostgreSQL の timestamp 型に書き換えるのが適切である．この型の書き換えを行った場合，関連する式も修正しなければならない場合がある．Oracle データベースでは DATE 型の値と数値型の値を「＋」「－」演算子により加減算できるが[b], PostgreSQL では TIMESTAMP 型の値と数値型の値をこれらの演算子により加減算できないためである．数値型の値に「day」などの単位を加えることによ

b) 数値型の値を日数として扱い，日付の加減算を行う．

り，PostgreSQL でもこれらの演算子を用いた同様の加減算が可能になる．

(4) **外部結合を含む SELECT 文の FOR UPDATE/SHARE 句**

PostgreSQL では，外部結合を含む SELECT 文において，NULL になる可能性がある表には FOR UPDATE/SHARE 句を適用できない．LOCK 文など別の方法によって行または表のロックを取得するように SQL コードを書き換える必要がある．

(5) **SELECT 文における ROWNUM 疑似列**

Oracle データベースでは，SELECT 文によって得られる結果の各行に連番を与えたい場合，ROWNUM 疑似列を用いることができる．PostgreSQL では ROWNUM 疑似列が使えないため，代わりに ROW_NUMBER 関数を用い，必要であれば AS 句によって「ROWNUM」という別名を与える必要がある．また，連番を与える行の順序が重要である場合は，さらに OVER 句によって順序を指定する必要がある．

5.4 稼働量削減効果の評価

ここまでに評価した 3 つのツールのうち，一部のみを用いた開発プロセス（以下，「開発プロセス I」と記す）と，全てを用いた開発プロセス（以下，「開発プロセス II と記す」と記す）で，稼働量がどれだけ削減されるかを試算した．ソフトウェア開発は設計，製造，試験などいくつかの工程に分かれるが，ここではツールが主として寄与する設計工程に焦点を当てて試算を行った．

試算はシステム甲の開発を例として行った．各開発プロセスの稼働量は人時単位で求めた．開発プロセス I では Ora2Pg のみを用いた開発を実際に行い，設計工程に要した稼働量を求めた．開発プロセス II については，本稿で評価した 3 つのツールを使用することに加えて，これらのツールを使用するための手順やツールが未対応の修正項目に関する知見がツール付属の文書として整備されていると仮定して，設計工程の稼働量を見積もった．

本来ならばツール以外の開発の条件は同一にして試算を行うべきである．しかし，実際には下記の通り，いくつかの条件が異なっている．そのため，試算の精度には改善の余地があることを付け加えておく．

- 移植先のアプリケーションサーバが異なる．開発プロセス I では JBoss，開発プロセス II では JBoss である．なお，これらはいずれも Java EE に準拠したアプリケーションサーバである．
- 移植先のデータベースサーバが異なる．開発プロセス I では Postgres-XC，開発プロセス II では PostgreSQL である．なお，Postgres-XC は多くの SQL について PostgreSQL と互換性がある．
- 開発者の技術水準が異なる．開発プロセス II には開発プロセス I よりも技術水準が平均的に高い開発者が携わっている．

試算結果について述べる．開発プロセス I の設計工程における稼働量の相対値を 1 とすると，開発プロセス II は 0.115 となった．すなわち，今回のシステム甲の開発の場合では，設計工程の稼働量が 88.5% 削減可能できると試算できた．

6. 考察

6.1 asupgrade に関する考察

v3 ツールの方が v2 ツールよりも編集作業量が大きかった．これは GlassFish 2.1.1 から同 3.1.2.2 で構成ファイルの XML 構造が一部変更になったため，その修正によるものと考える．

v2 ツール，v3 ツールとも，網羅性と自動化度はともに高かった．ツールが対応していない項目については，修正方法を文書にまとめ，ツールと併用することにより，円滑な移植が実現可能と考える．

6.2 Ora2Pg に関する考察

表については，名前が「rupd$_」または「mlog$_」から始まる表を書き換え対象から除くことによって自動化度を改善できると考える．これはツールの EXCLUDE オプションに適切な正規表現を指定することによって実現できると考える．

列については，NUMBER 型を numeric 型に書き換えることによって自動化度を改善できると考える．これはツールの DEFAULT_NUMERIC オプションに numeric を指定することによって実現できると考える．また，DEFAULT 式を持つ列，特にそれと同時に NOT NULL 制約を同時に持つ列については，ツールの出力結果を人手により確認し，必要であれば人手による修正を行うべきだと考える．

NUMBER(2) 型から smallint 型への書き換えなど，列の型の書き換えによって Oracle データベースで格納できなかった値が PostgreSQL で格納できるようになる場合がある．これが問題になる場合，アプリケーション側に値を検証する処理を追加するなどの対処が必要になると考える．

索引および順序については，ツールの出力結果をそのまま用いることができると考える．ただし，順序については，5.2 節の(4)にて述べたような最大値に関するデータベース間の仕様の差分がシステム上問題となる場合，再設計が必要になると考える．

表領域については，移植すべき対象をツールにより網羅的に出力できた．この出力結果を PostgreSQL の動作環境に応じて修正すれば，表領域の移植が可能である．ただし，Oracle データベースで指定できた細かなパラメータは PostgreSQL との仕様差分上出力されないため，場合によっては表領域の再設計が必要になると考える．

ユーザ定義型については，今回のような配列型の場合，

ツールが出力したものを用いるのではなく，列や変数の型宣言箇所で用いられているユーザ定義型を PostgreSQL の配列型に修正する方が，全体の修正が少なく済むと考える．ツールはそのような型を網羅的に抽出するための利用に留めるのが望ましいと考える．

ビュー，手続き，および関数については，他のツールを併用して移植するのが望ましいと考える．例えば，Ora2Pg が出力したこれらの SQL コードに対して db_syntax_diff を適用することにより，修正が必要な箇所をより網羅的に発見できるようになると期待できる．

増分更新を行う実体化ビューについては，Ora2Pg がビューとして出力する定義を利用しつつ，人手により再設計するのが望ましいと考える．例えば，[6]や[7]のように，詳細表に増分を追跡するためのトリガーを定義する方法がある．また，定期的に更新を行う実体化ビューの場合，外部プログラムを併用して更新方法を再設計する必要がある．これには，例えば，cron を用いる方法が考えられる．

シノニムやデータベースリンクなど，Ora2Pg が対応していないデータベース・オブジェクトがいくつか存在する．これらは別途対処する必要がある．

6.3 db_syntax_diff に関する考察

db_syntax_diff は非互換性が正規表現として記述された設定ファイルを用いて，SQL コード中の修正が必要な箇所を指摘するツールである．5.3 節にて示した類型(1)，(4)，および(5)については，特徴的な構文，キーワード，または文字列を含むため，正規表現で記述でき，ツールへの反映が容易であると考える．残る(2)および(3)については，空文字列リテラルおよび DATE 型であるような関数などの名前を正規表現として列挙することにより部分的な反映が可能と考える．しかし，文字列型や DATE 型の変数についても反映することは，SQL のキーワードや構文だけでなく意味も考慮する必要があるため，困難と考える．

db_syntax_diff が指摘する非互換性の一部は，PostgreSQL コミュニティが開発している Oracle データベース互換関数である orafce [8]を使用することによって，SQL コードを修正することなく移植可能である．orafce の使用を前提として，このような非互換性を db_syntax_diff の出力結果から取り除くことができれば，出力結果の確認と SQL コードの修正のための稼働量を削減することができると考える．

SQL コードを書き換えることによって，それらを使用するアプリケーションのソースコードも修正が必要な場合がある．これは Ora2Pg が行うデータベース・オブジェクトの定義の書き換えについても言える．これらの評価については今後の課題としたい．

6.4 稼働量削減効果に関する考察

一般に，ツールの効果はシステムの規模や特性などによって変化すると考えられる．したがって，今回ひとつだけのシステムで試算した削減効果をそのまま他のシステムに当てはめるのは適切ではない．評価対象のシステムを増やし，各システムの規模や特性その他の因子との相関を分析することによって，削減効果の精度を高めていくことが必要だと考える．

5.4 節にて述べた通り，削減効果を比較した開発プロセス間でツール以外の条件がいくつも異なるため，試算自体の精度にも改善の余地がある．また，削減効果のツールごとの分計についても，今回の評価では踏み込めていない．さらに，各ツールの設計以外の工程に対する削減効果についても，今回は評価できなかった．これらについては今後の課題としたい．

7. おわりに

本稿では Sun Java System Application Sever と Oracle データベースを対象として，アプリケーションを GlassFish に，データベースを PostgreSQL に移植するための 3 ツールを定量的および定性的に評価した．評価結果から，ツールの有効的な使用方法およびツールの改善方法を考察した．また，ツールによる稼働量削減効果を試算した．

今後は，6 節にて述べた課題の解決に取り組むとともに，ツールを適用するシステムを増やして，評価の精度を高めていく．また，ツールが指摘または自動的に修正できない非互換性については文書にまとめ，文書をツールと併用することによって移植をより容易なものとなるようにしていく．あわせて，それらの非互換性の中で，ツールに反映可能なものを順次反映し，ツールの能力を高めて行く．さらに，ツール使用時と未使用時で，移植したアプリケーションおよびデータベースの品質や性能などがどの程度変わるのか，ということについても評価を行っていく，

参考文献

1) To Upgrade From the Command Line (Sun GlassFish Enterprise Server v3 Upgrade Guide)
http://docs.oracle.com/cd/E19226-01/820-7698/gfcwm/
2) Ora2Pg
http://ora2pg.darold.net/
3) db_syntax_diff
https://github.com/db-syntax-diff/db_syntax_diff
4) プロジェクト GlassFish 向けマイグレーションツールの概要
https://wikis.oracle.com/display/GlassFish/M2GOverviewJa
5) Postgres Plus Advanced Server
http://www.sraoss.co.jp/prod_serv/package/postgresplus_as/
6) D. Chak: Materialized Views that Really Work, PGCon 2008
http://www.pgcon.org/2008/schedule/events/69.en.html
7) PostgreSQL/Materialized Views - Jonathan Gardner's Tech Wiki
http://tech.jonathangardner.net/wiki/PostgreSQL/Materialized_Views
8) Orafce - Oracle 互換関数
http://orafce.projects.pgfoundry.org/index-ja.html