東海大学紀要情報通信学部
Vol.8,No.1,2015,pp.30-37

論文

データベースの安全性のためのトリガを使用した表明の実装方式の提案とその自動生成ツールの開発

田中聖一[*1]，小林 洋[*2]

Trigger-Based Assertion for Database Safety : A Proposal of Implementation and Development of Generation Tool

by

Sei TANAKA[*1] and Hiromi KOBAYASHI[*2]

(received on Feb. 13, 2015 & accepted on Jun. 29, 2015)

あらまし

本論文では，最近のソフトウェア開発現場で浸透しつつある表明（assertion），つまりソフトウェア部品間の安全性（safety）を保障する記述の，UML/OCL で書かれた仕様からデータベース言語 SQL のトリガ（Trigger）への変換方式の提案と，これに基づき作成したツールのプロトタイプについて述べる．表明は，分散化されたソフトウェアの部品化つなぐための条件の記述であり，インターネットのように開いた分散系でのソフトウェアの安全性のために重要性が増している．従来，表明はアプリケーション側だけで実装が行われていたが，今後はデータベース側にも実装が望ましいと考えられるためこの方式を提案した．また，この実現性を実証するために，変換ツールの作成を行った．

Abstract

The description of assertions has been attracting attention recently for preventing programs from incorrect behavior in the field of software development. We propose a method of transformation from an OCL assertion to a trigger. Additionally, we developed a prototype transformation tool for PostgreSQL. OCL is a logical description language included in UML, which is used widely for the description of specifications. Triggers differ slightly from the vendor's dialect of SQL, but our transformation framework is applicable to others. Assertions must be implemented in a database as well as in program in open distributed systems such as Web application services.

キーワード: 表明，ソフトウェア開発，UML/OCL，トリガ，ソフトウェアの安全性
Keywords: *assertion, software development, UML/OCL, trigger, software safety*

1. はじめに

ソフトウェア設計において，契約による設計（Design by Contract）という観点から提唱されていた表明（assertion）の記述が，開発現場でも最近ようやく浸透しつつある．ここで言う表明とは，事前条件（precondition），事後条件（postcondition），および不変条件（invariant）と呼ばれるもので，事前条件はあるソフトウェア部品の入口で成立する条件，事後条件は出口で成立する条件，不変条件は常に成立する条件を言う．表明に書かれた条件に，ソフトウェア部品間でやり取りされるデータが適合しない場合には，エラー処理が行われることになる．表明を記述し実装することにより，ソフトウェアの振る舞いの安全性の向上を図ることができる．オブジェクト指向のソフトウェア開発では，UMLが広く用いられているが，表明を厳密に記述するためには論理を用いる必要がある．UMLでは，論理はOCL（Object Constraint Language）などで図式仕様に補足的に記述することになる[1-3]．表明はJava等のプログラム言語側で実装することが一般的であるが，インターネットのような開いた分散系の耐故障性の観点からは，表明をSQL等のデータベース言語側でも実装することが望ましいと考えられる．また，プログラムは劣化しないという観点から，表明はテスト工程でのみ実行され，実運用時には処理速度向上のためコメント化されるのが一般的であるが，開いた分散系においては実運用時の実装も必要であると考えられる．サーバ側で表明の実装をする場合，SQL92規格ではASSERTION文が制定されているが，製品レベルでは用いることが出来ない[4]．これは，従来の開発では，SQL側は出来るだけシンプルな記述とし，制約条件等はJava等のプログラム言語側で対処しようとしていたためである．現状，SQLでは簡単な表明はチェック（check）文でも表現可能だが，一般的にはストアドプロシージャ（stored procedure）の一種であるトリガ（trigger）を用いて実装を行う必要がある．トリガには，チェックのような宣言文よりも実行速度が速いという利点もある[5]．

*1 情報通信学研究科情報通信学専攻 修士課程
Graduate School of Information and Telecommunication Engineering, Course of Information and Telecommunication Engineering, Master's Program
*2 情報通信学部情報メディア学科 教授
School of Information and Telecommunication Engineering, Department of Information Media Technology, Professor

ところが，トリガの作成までプログラマに求めるのは，負担が大きすぎるように思われる．また，表明のような制約条件は，仕様書で論理を用いて厳密に記述した方が，誤解が少ない．そこで我々は，前研究[6-8]でOCLからトリガへの変換方法の提案とその自動変換ツールのプロトタイプの作成を行った．本研究では，形式化を更に進めると共に，操作性を考慮して変換用のツールの新たな作成を行った．更に，ArgoUML[9]やOmondo[10]等の広く普及しているUMLモデリングツールからクラス図を取り込むことが出来るようにXMI（XML Metadata Interchange）[11-12]形式のフォーマットを入力とすることが出来るようにした．変換用ツールは，Java言語を用いてPostgreSQLを対象に作成した．トリガは，SQL標準で定められているものの，SQLの製品によって多少異なる部分が存在するが，本論文で示す変換のフレームワークは，他の製品に対しても有用であると考えられる．以下，まず2．で関連研究について述べた後，3．では変換方式の概要，4．では表明の記述と変換手続きについて述べ，5.で開発したツール，6．で変換例と検証について示し，7．で考察を記す．

2．関連研究

OCLで記述した制約条件を，クライアント側のJava言語により実装する際の変換については，既にいくつかの研究が行われている．例えば，文献[13-14]では，OCLが付加されたUMLのクラス図において，クラス図から生成されるJavaコードに，OCLで記述した制約条件をJavaの注釈記述言語であるJML（Java Modeling Language）[15]に変換したものを組み合せて検証に用いている．また，Javaプログラムから表明を自動生成するツールも存在している．但し，その中で良く知られたツールであるDaikon[16-18]では，生成できる表明は変数間の関係が比較的単純なものに限られている．一方，OCLで記述した制約条件からサーバ側のSQL/Triggerへの変換については，データベースの管理におけるデータの完全性（integrity）という観点からの研究は行われているが[19-20]，ソフトウェア開発でのアプリケーションとデータベースの統合化という観点からの研究は行われていない．この理由は，従来のシステム開発では閉じた系を想定しており，プログラムモジュールの結びつきは予め全て明らかであり，また，プログラムは劣化することはないという観点から表明はプログラム言語側でのみ実装し，テスト工程でのみ実行すれば十分で，SQL側での実装は不要であると考えられていたためと思われる．最近，類似研究で，Dresden OCL22SQLツールキット[21]というのが開発されているが，これはOCLからSQLのテーブルに変換するもので，トリガは扱っていない．

3．変換方式の概要

前研究[6-8]において，我々は，OCLからトリガへの変換方法を提案し，変換用のプロトタイプツールの開発を行った．本研究では，この研究を更に進め，

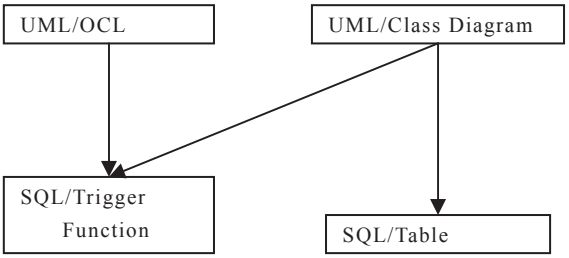


Fig.1 Transformation of assertion from OCL to trigger.

```
context class_name
assert_type : expression

where
assert_type = pre | post | inv

expression =
【expr_type1】
class_name.attri_name  operator1  value1
|【expr_type2】
class_name -> size() operator1 value1
|【expr_type3】
class_name -> select(class_name.attri_name
                operator2 value2) -> size() operator1 value1
|【expr_type4】
class_name -> isUnique(class_name.attri_name)

|【expr_type5】
class_name.attri_name
  -> excludes(class_name.attri_name  operator value) |
  -> includes(class_name.attri_name  operator value)
```

Fig.2 Template of OCL assertion.

まず，入出力の枠組み（template）とその間のマッピングによる手続きの形式化の拡張を行った．データベースはPostgreSQLを対象としている．トリガの記述は製品によって多少異なるところがあるが，入出力の枠組みとマッピングを多少変更すれば，他の製品の場合にも本方式は適用可能と考えられる．但し，PostgreSQLのトリガでは，関数も定義する必要があるため，Fig.1のように，クラス図とそれに補足して書かれたOCL文の表明から，トリガとその関数に変換を行う必要がある．なお，クラス図からSQLのテーブルへの変換も同時に行えるが，この変換方法については良く知られているので，本論文では触れないことにする．

4．表明の記述と変換手続き

4.1 OCLによる表明の記述

OCLは，UMLで制定されている論理仕様記述言語である．OCLではクラス図に記述されている型を使用して，クラス図に厳密な制約条件を付加することができる．本研究では，OCLによりソフトウェア開発で表明として用いられる事前条件，事後条件，および不変条件の記述を対象にする．UML等の図式仕様では，通常，表明はクラス図にコメントとして付加する．

本研究における，OCLによる表明の記述の枠組みをFig.2に示す．contextには，本方式ではクラスを記述する．assert_typeには，事前条件，事後条件，不変条件を表すpre，post，invのいずれかが記述される．expressionには，OCLの表明を式で表す．expressionの型はアプリケーションのドメインによって異なると考えられるが，今回対象とした履修登録システムのモデルにおいては，我々の検討結果では5つの型を見出すことが出来た．これらを，expr_type1，expr_type2，expr_type3，expr_type4，expr_type5と表すことにする．attri_nameは，class_nameの属性に対応する．operator1とoperator2には比較演算子を記述し，value1とvalue2には，定数を記述する．式のタイプについては，expr_type1はクラスが持つ属性値の制限をする表明である．本方式では，OCLの制約式において，クラス図から生成されたインスタンスの集合の検索を対象としているような場合には，コレクション演算を用いている．expr_type2は，コレクション演算のsize()演算を用いて，クラスのオブジェクト数を制限する表明である．expr_type3は，コレクション演算のsize()演算とselect演算を用いて条件を満たすクラスのオブジェクトの数を制限する表明である．size演算は，インスタンスの合計数を表す．select演算は，検索条件式が真となるインスタンスの集合を表す．expr_type4は，isUnique()演算を用いて，そのテーブルにおいて指定した属性の二重登録を防止する表明である．expr_type5は，excludes()またはincludes()演算を用いて，条件に一致したデータを制限する表明である．

4.2 トリガによる表明の記述

トリガ（trigger）はデータベースに何らかの変更処理（INSERT,UPDATE,DELETE）が加えられた時，その処理をきっかけにあらかじめ設定しておいた動作を実行する操作である．トリガ定義の記述では，対象となるテーブル，トリガの起動のきっかけとなる変更処理，トリガの処理内容，トリガの起動するタイミングを指定する．本研究で用いたPostgreSQLのトリガでは，トリガ部と関数部の2つの記述が必要である．トリガ部では処理のタイミングを記述し，関数部では具体的な処理の内容を記述する．Fig.3(a)にトリガによる表明の記述の枠組みを示す．枠組みの manipulation_type はトリガを起動させるSQL操作の指定で,INSERT,UPDATE,またはDELETEのいずれかが入る．trigger_type はトリガを起動するタイミングの指定で,BEFOREまたはAFTERが入り,各々SQL 操作が実行される前または後にトリガを起動することを表す．BEFORE と AFTER は表明では，それぞれ事前条件と事後条件に相当する．不変条件については，BEFORE と AFTER に同一のトリガを実装することにより疑似的に対処することにする．関数の枠組みについても，同様に形式化した．Fig.3(b)にその一部を示す．

```
CREATE TRIGGER trigger_name
manipulation_type
trigger_type
ON table_name
FOR EACH ROW
EXECUTE PROCEDURE
function_name();
where
  manipulation_type =
                DELETE | UPDATE | INSERT
  trigger_type = BEFORE | AFTER
```

(a) Template of trigger

```
CREATE FUNCTION  function_name()
RETURNS TRIGGER AS'
DECLARE
  count int;
  record_name RECORD;
cursor_name CURSOR
FOR SELECT * FROM table_name
WHERE variable1 operator_cur numeric1 ;
BEGIN
  OPEN cursor_name;
  count = 0;
  LOOP
    FETCH cursor_name
    INTO record_name;
    IF NOT FOUND THEN
      EXIT;
    END IF;
    count = count + 1;
  END LOOP;
  IF count operator_if numeric2 THEN
    RETURN NEW;
  END IF;
  CLOSE cursor_name;
  RETURN NULL;
END; '
LANGUAGE 'PLPGSQL';
```

(b) Template of function (part)

Fig.3 Template of trigger and its function.

4.3 変換手続き

Fig.2で示したOCLの枠組みに記述されたデータを，Fig.3 に示したトリガと関数の枠組みに記述されたデータに変換する手続きの一部を Fig.4 に示す．Fig.4 のトリガと関数に使用される triggar_name, function_name,および cursor_name には任意の名前を記述する．Fig.4(a)は OCL からトリガへの変換手続きであるマッピングである．表明の型を示すassert_typeにより，トリガ起動のタイミングを表すtrigger_typeが決定する．表明の型が事前条件(pre)ならば trigger_type は BEFORE となり，事後条件

（post）ならば trigger_type は AFTER となる．不変条件（inv）である場合には，BEFORE と AFTER の両方を作成する．また，OCL のクラス名をトリガのテーブル名へ代入する． Fig.4(b)は OCL から関数への変換手続きであり，Fig.4(c)はクラス図から関数へ OCL を利用して行う変換手続きである．

5. 開発したツール

4.で示した変換手続きを用いて，OCL からトリガと関数へ自動変換する開発支援ツールのプロトタイプを，Java により作成した．このツールでは，入力は外部の UML モデリングツール[9-10]等からの XMI 形式のデータを想定している．但し，ツールによってクラス図の XMI データに異なる部分があるため，今回は共通の基本的な部分の記述に限定してある．また，OCL についても XMI 形式のデータによる入力が望ましいが，OCL については XMI の標準化が進んでいないので，今回はエディタによる直接入力またはテキストファイルの読み込みにより行っている．このツールの構成は，以下のとおりである．

・クラス図パーサ（Class Diagram Parser）
・クラス図テーブル変換器（Class-Table Converter）
・OCL 表明エディタ（OCL Assertion Editor）
・OCL 構文解析器（OCL Syntax Checker）
・OCL－トリガ変換器（OCL-Trigger Translator）

上記において，クラス図パーサには SAX を使用しており，OCL 構文解析器には JavaCC を使用している．

Fig.5 にツールによる変換手続きの全体の流れを示す．Fig.6(a)は XMI 形式のクラス図のファイルの入力画面で，(b)は OCL－トリガ変換器の画面を示す．変換は，Fig.5 で示した流れに従って，次のように行われる．Fig.6(a)の画面からクラス図が入力されると，パーサを通してチェックされた後，クラス図テーブル変換器によりテーブルへの変換が行われる．一方，表明は Fig.6(b)の画面左側のテキストボックスに Fig.2 の OCL の枠組みに従って直接入力するか，または左上の Opentext ボタンによるテキストファイルの読み込みにより入力が行われ，続いて構文解析が行われる．構文解析では，条件に and や or の有無等の判別を行った後，条件式のパターンの判別を行う．この際，テンプレートに沿わない記述がされていた場合にはエラーを返す．次に，任意のトリガ名と関数名を左下の入力フォームに記述した後，中央の Generate ボタンを押すと OCL トリガ変換器により，OCL とテーブルからトリガと関数に変換が行われ画面右側に表示され，右下 Save ボタンを押すことにより，テキストファイルに保存することが出来る．

```
IF(OCL)assert_type = pre
    THEN(Trigger)trigger_type = BEFORE
ELSE IF (OCL)assert_type = post
    THEN(Trigger)trigger_type = AFTER
ELSE IF(OCL)assert_type = inv
    THEN(Trigger)trigger_type = BEFORE
                                and AFTER
END IF
(Trigger)table_name=(OCL)class_name
```

(a)　Mapping form OCL to TRIGGER

```
(Function)table_name=(OCL)class_name
(Function)operator_if = (OCL)operator1
(Function)numeric2      = (OCL)value1
IF(OCL)expression is【expr_type1】
     THEN (Function)end_loop2 = “END LOOP”
      AND (Function)end_loop1 = “”
ELSE IF expression is【expr_type2】or【expr_type3】
    THEN (Function)end_loop1 = “END LOOP”
     AND (Function)end_loop2 = “”
ENDIF
IF(OCL)expression is【expr_type3】
    THEN (Function)variable1 = (OCL)attri_name
     AND (Function)operator_cur = (OCL)operator2
     AND (Function)numeric1 = (OCL) value2
  ELSE IF expression is【expr_type2】or【expr_type3】
      (Function)variable1 = “”
      AND (Function)operator_cur = “”
      AND (Function)numeric1 = “”
END IF
```

(b)　Mapping from OCL to Function

```
IF(OCL)expression is 【expr_type1】
      AND (OCL)attri_name = (Class)attri_name
   THEN  (Function) var_type = (Class)attri_type
     AND  (Function)key = (Class)attri_name
ELSE IF(OCL)expression is【expr_type2】
                                  or【expr_type3】
   THEN  (Function)var_type = (Class)key_type
    AND  (Function) key= (Class ) primary_key
END IF
```

(c)　Mapping from Class to Function Using OCL

Fig.4 Transformation from OCL and class to trigger and function (part).

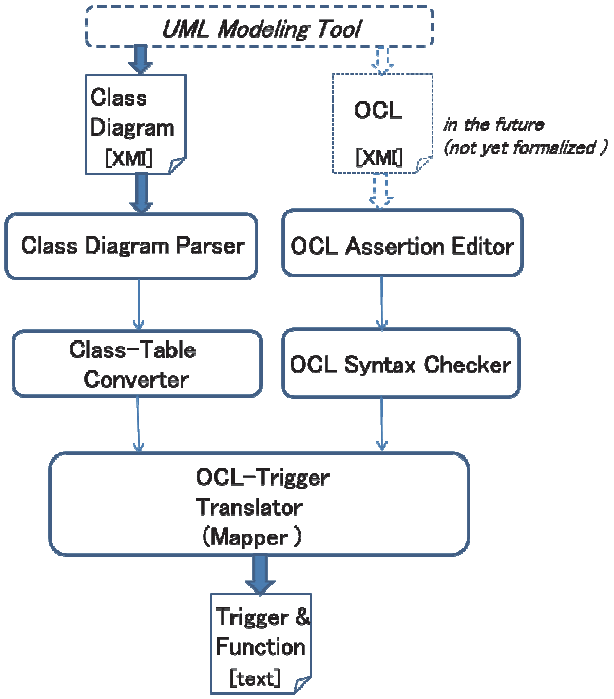
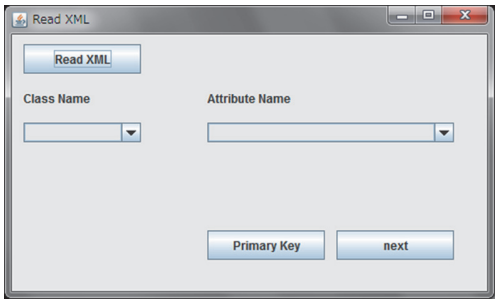
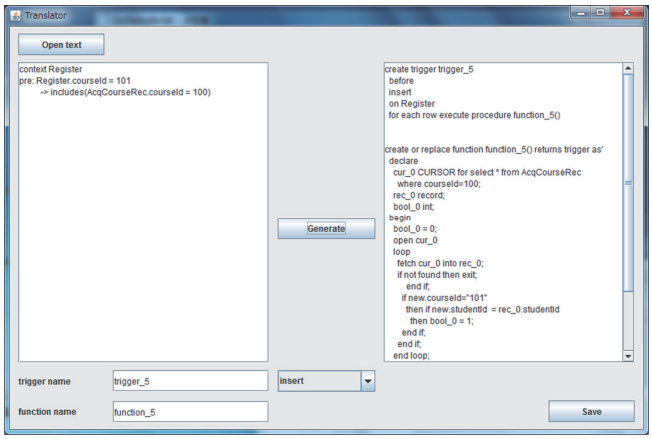


Fig.5 Activity of our tool.



(a) GUI of class attributes



(b) GUI of OCL-trigger translator

Fig.6 Snapshots of OCL-trigger translator.

6. 変換例と検証

ツールを用いた変換例と，その動作確認の検証について示す．先ず，大学の履修登録システムをモデル化したものを基に，表明の OCL での記述例を示し，次に，トリガへの変換例を示す．履修システムの一部をモデル化したものを Fig.7 に示す．(a)はクラス図，(b)は(a)を元にバックマン図(Bachmann diagram)に変換したものでデータベースでのテーブル間の関係を示す．(b)で，Student テーブルは，属性名として学生番号を表す studentId，学生名を表す studentName，および現在のセメスタを表す prSemester を持つ．Course テーブルは，属性名として授業番号を表す courseId，授業名を表す courseName，履修可能な最小セメスタを表す minSemester，および授業の開講曜日・時限を表す opPeriod を持つ．Register テーブルは，各学生の現在のセメスタの履修登録状況を表し，AcqCourseRec (Acquired Course Record) テーブルは，各学生が今までに履修した授業履歴の中から修得済みのもののみ抽出したものとする．この図に付加する表明を OCL で記述したものの例を次に示す．

(a)[type1]属性値の制限

```
Context Student
pre: Student.prSemester <= 16
```

Student の現在のセメスタ prSemester は 16 以下でなければならない．

(b)[type2]インスタンス数の制限

```
Context Student
inv: Student -> size() <= 50
```

Student のインスタンス数，つまり人数は 50 人以下でなければならない．

(c)[type3]条件に一致したインスタンス数の制限

```
Context Register
pre : Register ->
  select(Register.courseId = 100 ) -> size() <= 50
```

Register において courseId が 100 の授業のインスタンス数は 50 以下，つまり 50 人までしか受けることができない．

(d)[type4]二重登録の防止

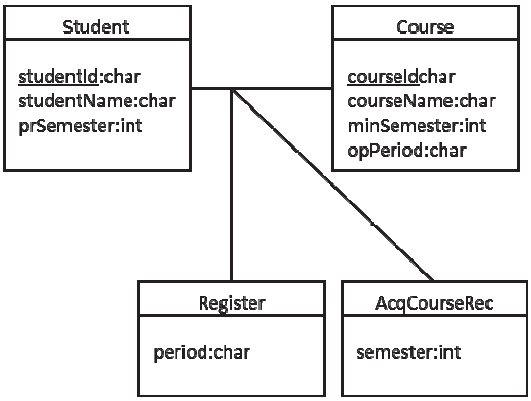
```
Context Course
pre: Course -> isUnique(Course.courseName)
```

courseName の二重登録の防止．

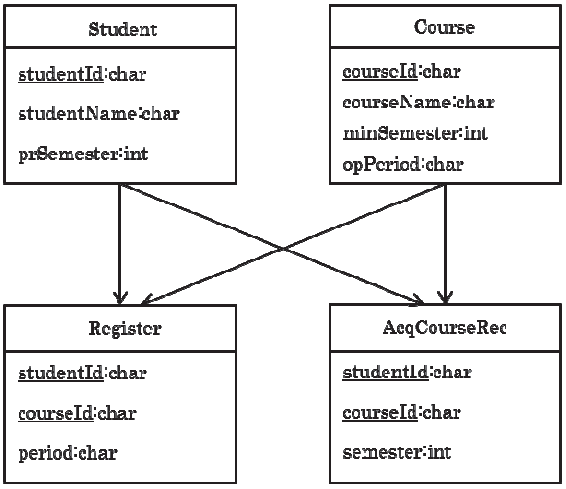
(e)[type5]条件に一致したデータの制限

```
 Context Register
pre : Register.courseId = 101
            -> includes(AcqCourseRec.courseId = 100)
```

courseId が 101 の授業を履修するためには courseId が 100 の授業を先に修了していなければならない．



(a) class diagram



(b) Bachman diagram

Fig.7 Course registration model.

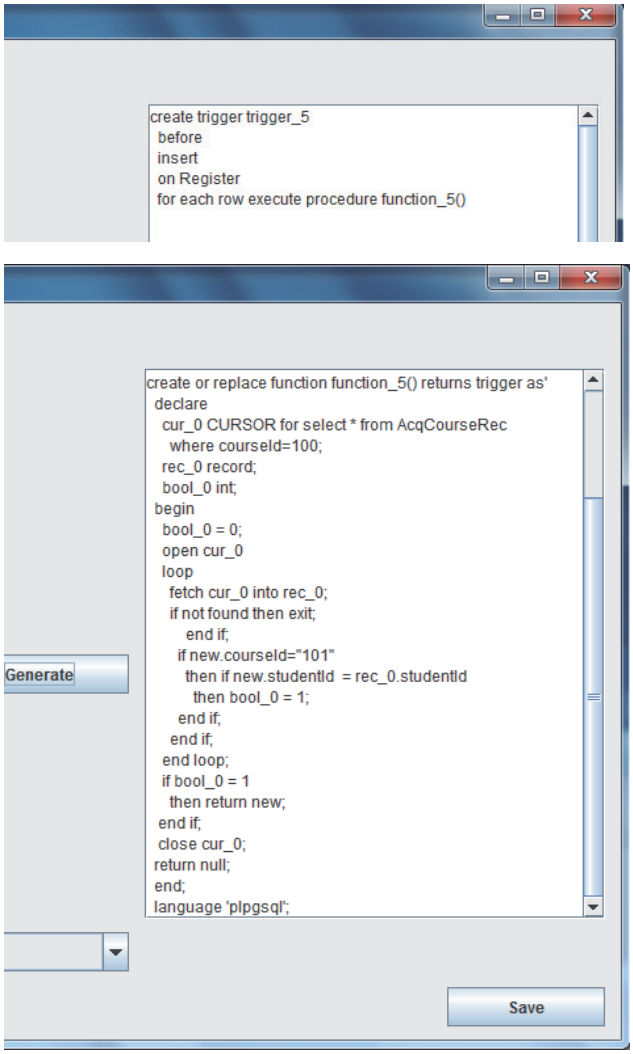


Fig.8 Example of a transformation result.

以上の5つのタイプの表明の OCL 記述について，我々の開発したツールを用いて，トリガ記述と関数に変換することが出来た．例えば(e)についての変換結果は Fig.8のようになる． Fig.8では，上4行がトリガ，その下にスクロールさせて示しているのが関数である．

次に，以上の5つのタイプのトリガと関数が表明として正しく働くことを PostgreSQL に実装して検証を行った．例えば(e)では，courseId が101の授業の履修の前に，courseId が100の授業を履修していれば履修登録が行われ，そうでなければ履修登録が行われないことの確認を行った．条件を満足しない場合の処理については，現在は処理を行わないだけにしているが，エラー処理の方法については，今後，検討の余地がある．

7. 考察

従来のシステム開発は，以下のような観点の基に行われていた．

（C-1）システム開発は閉じた系を扱っており，開発するプログラムモジュールは，どのようなプログラムモジュールと結びつくか予め解っている．

（C-2）プログラムは，ハードウェアと異なり，一度作られると劣化することはない．

（C-3）SQL 等のデータベース言語側は出来るだけシンプルな記述とし，表明等の制約条件はプログラム言語側に実装する．

そのため，表明はプログラム言語側に実装され，テスト工程でのみ実行され，実運用時には処理速度の向上のため，通常，コメント化されていた．

一方，インターネットのように開いた系での開発においては，以下のようなことが言える．

（F-1）開いた系においては，開発するプログラムモジュールは，どのようなプログラムモジュールと結びつくか，予測しきれない．

（F-2）プログラムは，マルウェア等により，書き換えられる恐れがある．

そのため，表明はプログラム言語側（クライアント側又はアプリケーションサーバ側）のみならず，データベース言語側（サーバ側又はデータベースサーバ側）にも実装し，テスト工程のみならず，実運用時にも実行するのが望ましいと考えられる．処理速度については，現在のコンピュータ等ハードウェアの性能向上は著しいので，将来的にはあまり問題にならなくなって来るものと思われる．

現状では SQL 側で表明を実装する場合，SQL92 では ASEERTION 文が制定されているが，上記の（C-3）のように，従来は SQL 側では表明は不要と考えられているためか，どの SQL 製品にもサポートされていない．従って，現状では SQL 側で表明を実装するには，簡単なものは CHECK 文で可能であるが，やや複雑なものになるとトリガを使わざるを得ない．表明を実装する際の，従来のクライアント側のプログラムにおける Java での assert 関数を使用する方法とサーバ側でのトリガ，CHECK 文，および ASSERTION 文を使用する方法についてまとめたものを Table 1 に示す．

Table 1 Comparison of assertion implementation.

	Java assert関数	trigger	CHECK文	ASSERTION文
記述場所	クライアント側 又は アプリケーションサーバ側	サーバ側 又は データベースサーバ側		
記述方法	手続き型		宣言型	
実行速度	速い		遅い	
表現力	複雑な関係も記述可能		単純な関係のみ記述可能	複雑な関係も記述可能
記述の標準化	Java言語標準	SQL標準 但し，製品により異なる	SQL標準	SQL標準 但し，未実装

8. おわりに

本論文では，UML のクラス図に OCL で記述された表明を，現状の DBMS で実現するために，トリガを用いて実装する方式を提案し，まず，OCL からトリガとその関数を生成する方式について，入出力の枠組みとマッピングにより処理手続きの形式化を行った．前研究では，形式化が不十分であり，開発した変換ツールのプロトタイプも操作性が悪く，機能が不十分であったので，今回新たにツールの開発を，Java 言語を用いて行った．対象とした DBMS は PostgreSQL である．トリガは DBMS により多少異なるところがあるが，他の DBMS であっても，本論文で示した方式の入出力の枠組みとマッピングを変更すれば実現可能であると考えられる．また，将来 ASSERTION 文が商用データベースでサポートされるようになったとしても，表現力の柔軟性と実行速度の速さという点でトリガは優れており，本研究のような手法は有用であろうと考えられる．

現状の開発においては，表明はプログラム言語側に記述し，SQL はシンプルにするのが一般的であるが，今後はデータベースサーバ側にもストアドプロシージャとして表明の記述が，インターネットのように開いた分散系の耐故障性の点からは，必要であろうと思われる．また，プログラムは劣化することがないという信頼性の立場のみでなく，マルウェアによりプログラムは書き換えられるというセキュリティの面も考慮して，表明を従来のようにテスト工程時のみでなく，実運用時にも実行する必要があると考えられる．

表明については，今回対象とした履修登録システムのモデルにおいては，現状5パターンを見出すことが出来た．表明は，ドメインに依存するところがあるので，今後の課題としては，とりあえず大学教育分野に限定して，表明のパターン化とそれに基づくツールの拡張があると考えられる．更には，業務系システムにおいては，表明について共通点も多いと思われるので，業務系のその他のシステムでの検討も必要があると考えている．今後，新たなパターンが見つかった場合でも，本手法のマッピングによる変換部分の追加・修正により，ツールへの追加が可能であると考えられる．なお，トリガはテーブルにおけるデータの修正により引き起こされるものであるので，その副作用についても，今後検討する必要があると考えられる．

参考文献

1) T. Clark: Typechecking UML Static Model, LNCS1723, Proc. UML’99, pp. 503-517, 1999.
2) M. Bidoit, R. Hennicker, F. Tort, and M. Wirsing: Correct realizations of interface constraints with OCL, LNCS1723, Proc. UML’99, pp. 399-415, 1999.
3) L.A. Clarke and D.S. Rosenblum: Runtime assertion checking, IEEE Computer, Vol.41, No.2, p.48, 2008.
4) R. T. Snodgrass: Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, 2000.
5) H. Decker, D. Martinenghi, and H. Christiansen, Integrity checking and maintenance in relational and deductive databases and beyond, Intelligent Databases: Technologies and Applications, Idea Group Publishing, 2006, pp. 238-285.
6) 黒澤慎太郎,小林洋: 表明のUML/OCLからSQLへの変換, FIT2008, 第1分冊, pp.123-124, 2008.
7) S.Kurosawa and H.Kobayashi: Transformation of UML/OCL Assertion to SQL/Trigger in Software Development, Proc. 2011 International Conference on Computer, ICCESSE2011 ,pp.2654-2659, 2011.
8) S. Tanaka, S. Kurosawa, and H. Kobayashi: Server Side Assertion Using SQL/Trigger from UML/OCL in Software Development, Proc. IEEE 2012 Pacific Rim International Symposium on Dependable Computing, 2012.
9) ArgoUML:http://argouml.tigris.org
10) Omondo: http://www.omondo.com
11) OMG MOF 2 XMI Mapping Specification:

http://www.omg.org/spec/XMI.
12) ISO/IEC19503:2005 XML Metadata Interchange: http://www.iso.org
13) A. Hamie: Translating the object constraint language into the Java modeling language, Proc. 2004 ACM symposium on applied computing (SAC2004), pp.1531- 1535, 2004.
14) R. Moiseev and A. Russo: Implementing OCL to JML translation tool, IEICE Technical Report, SS2006-58, pp.13-17, 2006.
15) G. T. Leavens, A. L. Baker, and C. Ruby: JML: A notation for detailed design, Behavioral Specifications of Businesses and Systems, pp.175-188, Kluwer Academic Publishers, 1999.
16) M.D. Ernst, J. Cockrell, W.G. Griswold, and D. Notkin: Dynamically discovering likely program invariants to support program evolution, IEEE Trans. Software Engineering, Vol.27, No.2, pp.99-123, 2001.
17) M.D. Ernst, J.H. Perkins, P.J. Guo, S. Mccamant, C. Pacheon, M.S. Tschantz, and C. Xiao: The daikon system for dynamic detection of likely invariants, Science of Computer Programming, vol.69, no.1-3, pp.35-45,2007.
18) J.W. Nimmer and M.D. Ernst: Automatic generation of program specifications, Proc. 2002 Int'l Symposium on Software Testing and Analysis (ISSTA), pp.232-242, 2002.
19) M. Badawy and K. Richta: Deriving triggers from UML/OCL specification, Information Systems Development: Advances in Methodologies, Components, and Management, pp.305-315,Springer,2002.
20) H.T.Al-Jumaily, D.Cuadra, and P.Martinez: OCL2Trigger: Deriving active mechanisms for relational databases using Model-Driven Architecture, Journal of Systems and Software, pp.2299-2314, Elsevier, 2008.
21) DresdenToolkit: http://dresden-ocl.sourceforge.net/usage/ocl22sql/.