PUSH型とPULL型のプロトタイプを自動生成可能なアーキテクチャ設計ツール

太田 陽一朗[1,a)]　新田 直也[1,b)]

概要：ソフトウェア開発において，コンポーネント間のデータの転送を PUSH 型で行うか PULL 型で行うかは非常に重要な要素である．データの転送方式の選択はシステム全体の構造とパフォーマンスに大きな影響を与える．そのため，アーキテクチャ設計時に適切なデータ転送方式の選択を支援することが求められる．本研究では，アーキテクチャ記述言語で定義されたモデルに基づいて，PULL 型と PUSH 型の Java プロトタイプを自動生成するツールを開発し，簡単なモデルでパフォーマンス評価を行った．

1. はじめに

ソフトウェアアーキテクチャを構成する上で，コンポーネント間のデータの転送を PUSH 型で行うか PULL 型で行うかは非常に重要な要素である [1], [2]．**PUSH 型のデータ転送**とは，あるソフトウェアコンポーネントから別のソフトウェアコンポーネントに制御が移るときに，制御の移動と同じ向きにデータを転送する方法であり，**PULL 型のデータ転送**とは制御の移動と逆向きにデータを転送する方法である．例えばコンポーネント間でのメソッド呼び出しを考えたとき，引数を使ったデータ転送が PUSH 型に，戻り値を使ったデータ転送が PULL 型に相当する．本研究ではデータの転送方式を PUSH 型と PULL 型の間で双方向に変更可能なアーキテクチャレベルのリファクタリングを提案し，そのようなリファクタリングを支援するための枠組みの構築を目指す．一般にリファクタリングとはソフトウェアの外的な振る舞いを変えることなく，内部構造を改善する枠組みのことであり，ここではデータの転送方式が，改善すべき内部構造に相当する．データの転送方式の変更によって，データの使用量や実行効率などの非機能的要件の改善を図ることができる．一方，データの転送方式を変更しても変わらない振る舞いに相当するのが，データ間の依存関係である.. ただし，システムが持つデータには利用者から観測可能なものと観測不可能なものがある．したがって，リファクタリングの前後で変わらないことを保証しなければならないのは，外部から観測可能なデータ間の依存関係ということになる．そこで本稿では，システム全体のデータ間の依存関係を形式的に定義できるアーキテクチャモデルを導入する．本アーキテクチャモデルは，他の多くのアーキテクチャモデル [3], [4] がプロセスを中心に構成されているのに対し，データを中心に構成されている点が特徴である．入力されたアーキテクチャモデルをグラフを用いて表現し，ユーザによって選択されたデータ転送方式の情報から，Java プログラムのプロトタイプを自動生成するツールを開発した (図 1 参照)．生成されたプロトタイプを実行してパフォーマンス評価を行った．

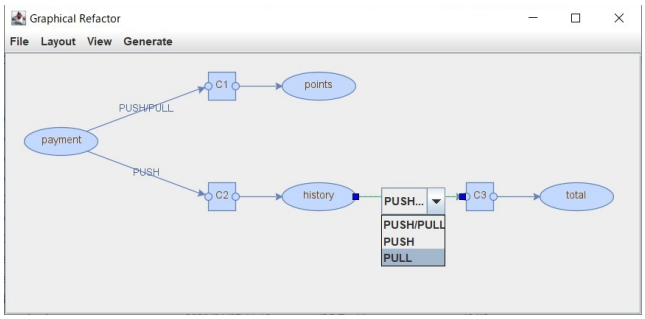

図 1　グラフベースドアーキテクチャ設計ツール

2. リソースに基づくアーキテクチャ記述言語

例として，POS システムのアーキテクチャモデルを図 2 に示す．本研究では，外部から観測可能な見掛け上のデータを**リソース**と呼ぶ．各リソースは**チャンネル**を通じて繋がっており，データはチャンネル内を入力側のリソースから出力側のリソースに向かって流れる．チャンネル C1 を例にとるなら，payment は入力側, points は出力側のリソースになる．チャンネル CIO は外部からの入力を受け

1　甲南大学大学院 自然科学研究科
Graduate School of Natural Science, Konan University
a)　m1924003@s.konan-u.ac.jp
b)　n-nitta@konan-u.ac.jp

```
channel CIO {
   out payment(p:Int, purchase(x:Int)) == x
}
channel C1 {
   in payment(p1, update1(y)) == y
   out points(l:Int, update1(y)) == floor(y * 0.05)
}
channel C2 {
  in payment(p, update2(z)) == z
  out history(h:List, update2(z)) == cons(z, h)
}
channel C3 {
  in history(h, update3(u)) == u
  out total(t:Int, update3(u)) == sum(u)
}
```

図 2 POS システムのアーキテクチャモデル

付ける特殊なチャンネルであり，出力側リソースのみが指定される．単独のチャンネルに複数の入力側リソースや出力側リソースを指定することも可能である．

3. グラフベースドアーキテクチャ設計ツール

アーキテクチャ設計ツールとして，図 1 のようなツールを開発した．このツールは，図 2 のように定義されたアーキテクチャモデルを入力として受け取り，解析を行って，リソースとチャンネルの関係を図 1 のようにグラフを用いて表現する．さらに，ユーザーがプルダウンメニューから選択したデータ転送方式を元に，実行可能な Java のプロトタイプを自動生成する．グラフベース UI は，JGraphX を使用して実装した．このツールを使用して，図 2 のモデルから，自動生成したプロトタイプの一部を図 3 に示す．

4. プロトタイプの速度評価

3 章で自動生成した PUSH 型優先と PULL 型優先のプロトタイプの実行速度評価を行った．各メソッドについて，メソッド呼び出しの 10,000 回の反復を 5 回実行する時間を測定し，平均値を算出した．その結果を図 4 に示す．予想どおり，PUSH 型の実装に関しては，すべての PUSH 型の転送が実行される purchase() の計算時間は非常に長く，PULL 型の実装に関しては，最新の状態を再計算するゲッターメソッドが，PUSH 型と比べて長かった．

5. おわりに

本稿では，データの転送方法の選択を支援するアーキテクチャ設計手法を提案し，観測可能なデータの振る舞いを表すことができる代数的アーキテクチャ記述言語を提示した．また，アーキテクチャ設計手法を支援するグラフベースのツールを開発し，ツールによって生成された PULL 型および PUSH 型プロトタイプの計算パフォーマンスを評価した． 結果として，全体的な構造と計算パフォーマンス

```
public class Payment {
   private int payment = 0;
   private Points points = null;
   private History history = null;
   public void purchase(int payment) {
      this.payment = payment;
      points.update(payment);
      history.update(payment);
   }
   public int getPayment() {
      return payment;
   }
}
public class Points {
   private int points = 0;
   public void update(int payment) {
      points = (int)((double) payment * 0.05);
   }
   public int getPoints() {
      return points;
   }
}
```

図 3 POS システムの PUSH 型優先のプロトタイプの一部

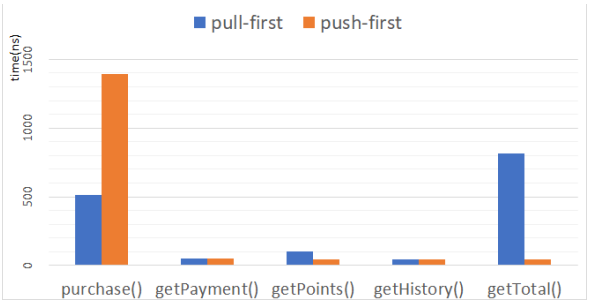

図 4 プロトタイプの速度評価結果

の両方がデータの転送方法の選択によって影響を受けることを示した．今後の課題として，より大規模なアプリケーションに適用できるようにアーキテクチャモデルを拡張したいと考えている．

参考文献

[1] M. Besta, M. Podstawski, L. Groner, E. Solomonik and T. Hoefler. To push or to pull: on reducing communication and synchronization in graph computations. In *Proc. of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. pp. 93–104, 2017.

[2] Y. Zhao. *A model of computation with push and pull processing*. Technical Report UCB/ERL M03/51. EECS Department, University of California, Berkeley, 2003.

[3] U. Klein and K. S. Namjoshi. Formalization and automated verification of RESTful behavior. In *Proc. of the 23rd international conference on Computer aided verification (CAV' 11)*, pp. 541–556, 2011.

[4] P. Pelliccione, P. Inverardi and H. Muccini. Charmy: a framework for designing and verifying architectural specifications. *IEEE Trans. on Software Engineering* Vol. 35, No. 3, pp. 325–346, 2009.