アスペクト指向に基づくソフトウェア開発支援に関する研究
– リフレクションを用いて –

2005MT084 小川 沙織
指導教員 蜂巣 吉成

1 はじめに

本研究室では組込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャスタイル (E-AoSAS++)[1] が提案されている．E-AoSAS++ は組込みソフトウェアを並行状態遷移機械 (CSTM) の集合と規定する．E-AoSAS++ に基づくソフトウェア開発では関心事をアスペクトとしてモジュール化し，再利用性が高いソフトウェアを目指している．また，E-AoSAS++ では開発労力削減のために，アーキテクチャからコードを自動生成する生成系が提案されている．

しかし，E-AoSAS++ ではクラスのインスタンスが複数存在する場合の考察が不十分であった．現在，複数のインスタンスを扱えるアーキテクチャが新たに提案され，そのアーキテクチャに基づきプラットフォームコードが設計，実現されている．しかし，生成系は，従来の E-AoSAS++ に基づき作成されているので，新しいアーキテクチャに対応できていない．

本研究の目的は，複数のインスタンスを扱えるプラットフォームコードをリフレクションを用いて設計することである．リフレクションを用いることでプラットフォームコードに生成系も定義することができ，生成系は必要なくなる．また，コードを整理し，コードの実現手順を明確にしたことで，保守がしやすいコードを実現することができると考える．

2 E-AoSAS++

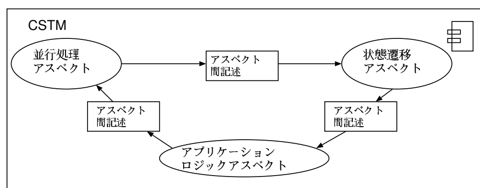

図 1 CSTM の構造

E-AoSAS++ では組込みソフトウェアを並行状態遷移機械 (CSTM) の集合と規定し，UML の図式表現を用いてソフトウェアを設計する．各 CSTM はイベントを受けることで動作し，複数の CSTM が連動することで，組込みソフトウェアの機能を実現している．図 1 に CSTM の構造を示す．並行処理アスペクトでは，CSTM に通知されるイベント管理や並行動作を扱う．状態遷移アスペクトでは，イベントに対応した状態遷移を扱う．アプリケーションロジックアスペクトでは，状態遷移に伴う CSTM が保持するデータに関する処理を扱う．アスペクト間記述は，各アスペクトを関連付ける．

新たに提案されたアーキテクチャでは，複数のインスタンスの構造や関連を管理する InstanceTable をインスタンス処理アスペクトとして追加した．実行時に各クラスのインスタンスを生成，登録し，各インスタンスに情報を提供する．

3 リフレクション

リフレクションは，プログラム中から，そのプログラム自身をデータとして取り扱い，計算の対象にする．Java 標準のリフレクション機能では，数値や文字列だけでなく，クラスやメソッドの定義もオブジェクトとして表され，データとして扱い，クラス内を内観し，メソッドなどを取得し，実行することができる．また，Java 標準のリフレクション機能ではないが，構造リフレクションという機能が提案されている．構造リフレクションとは，プログラム中のクラスやメソッドなどのデータ構造の定義を必要に応じて変更できる機能である．構造リフレクションの機能を提供するクラスライブラリ Javassist[2] は指示された変更内容を実行時にバイトコードに変換し，変換されたバイトコードをプログラムコードに反映する．

このようなリフレクション機能を用いると，プラットフォームコードが自身を書換え，内観することなどができるので，プラットフォームコード自身が生成系となる．

4 リフレクションを用いたプラットフォームコードの設計

4.1 概要

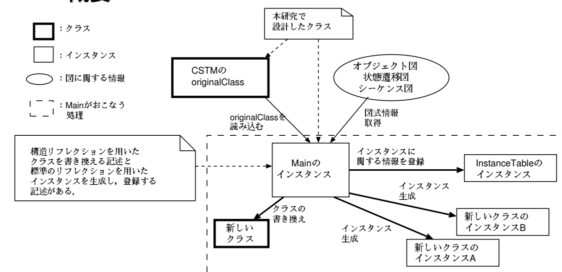

図 2 概要

図 2 に本研究で提案するリフレクションを用いたプラットフォームコードの概要を示す．リフレクションを用いた CSTM のクラス originalClass とクラスの書換えとインスタンスの生成，登録をするクラス Main を設ける．Main は図式表現から情報を取得し，それを基に originalClass を書換える．次に，書換えられたクラスのインスタンスを生成し，インスタンスに関する情報を InstanceTable に登録する．Main はこの動作を全ての図式情報に対して繰り返す．

AspectJ を用いたプラットフォームコードを基に，リフレクションを用いたプラットフォームコードを設計する．実行時の効率，不具合の発生を考慮して，構造リフ

レクションによるクラスの書換えは最小限に抑える．
書換えるコードは 2 種類ある．
1 つ目は，中身は定型コードで名前のみ異なるメソッドである．メソッド内では，Java 標準のリフレクション機能を用いて名前を基に呼び出すメソッドを動的に取得し，呼び出す．名前は Java クラスの field に持たせ，実行時に field から名前を取得する．
2 つ目は，クラス名の変更とメソッドを追加するというコード自体が異なるコードの書換えである．これらは Java 標準のリフレクション機能では実現できないので，構造リフレクションを用いる．
インスタンス生成は，図式情報を基に Java 標準のリフレクション機能で生成することができる．

4.2 図式情報

表 1 に各図から取得できる情報を示す．オブジェクト図，状態遷移図，アクションを表すシーケンス図からプラットフォームコードに必要な図式情報を取得する．

表 1　図からの取得情報

図式	取得情報
オブジェクト図	インスタンス名，インスタンスの数
状態遷移図	CSTM 名，遷移前の状態名，遷移後の状態名，状態の数，イベント名，イベントの数，アクション名
シーケンス図	アクション名，イベント名，イベント通知先の CSTM 名

4.3 Java 標準のリフレクション機能を用いたプラットフォームコードの設計

クラスの書換えをおこなう前のクラス設計の例として，状態遷移アスペクトの originalStateTransition クラスを説明する．このクラスのコードを図 3 に示す．各アスペクトを構成するクラスの汎用的なコードを API として用意し，利用するサブクラスとして定義した．設計方針に基づき，field に初期状態名を持たせた．メソッド内では，呼び出すメソッド名からそのメソッドを持っているオブジェクトを取得し，呼び出している．

4.4 構造リフレクションを用いたプラットフォームコードの設計

クラス書換えの例として状態遷移アスペクトの originalStateTransition クラスについて説明する．図 3 に originalStateTransition クラスの書換えの流れを示す．このクラスを書換えるためには，CSTM 名と初期状態名，イベント数，アクション名が必要である．クラス名を変更し，初期状態名やイベント名などを取得し，イベント数分メソッドを追加する．
全てのクラスの書き換えが終了するとインスタンス生成をする．インスタンス生成には，インスタンス名とインスタンスの数も必要になる．インスタンス生成時に InstanceTable の登録もおこなう．

5 考察

従来の生成系では，コードや図式表現に修正がある場合，実行されるプラットフォームコードを修正し，それに対応するコード生成系も修正する必要がある．しかし，リフレクションを用いることでコードに生成系も定義する

```
import java.lang.reflect.* ;
        :
public class cstm_nameStateTransition extends StateTransition implements State {
        private State currentState ;
        String firsrStateName ;
        String evName ;
        public static void setFirstStateName(String cstm_name, String state_name) {
                firstStateName = cstm_name + "State" + state_name ;
        }
        public static String getFirstStateName() {
                return firstStateName ;
        }
        public static void setEventName(String event_name) {
                evName = event_name ;
        }
        public static void getEventName() {
                return evName ;
        }

        public cstm_nameStateTransition () {
                String first = getFirstStateName() ;
                 Class<?> classFor = Class.forName( first ) ;
                 State currentState = (State) classFor.newInstance () ;
        }
        public void trans (Event ev) {
                 Class<?> classFor = this.getClass() ;
                 Method [ ] M = this.getMethods () ;
                 try{
                    for ( i = 0 ; i < M.length ; i++ ) {
                         Method m = M[ i ] ;
                         String name = m.getName () ;
                         if ( name.startWith("Event_" + getEventName()))
                              m.invoke( this ) ;
                                        :
                    }
         }

        public void Event_evName() {
               Class<?> currentStateclass = currentState.getClass() ;
               ArrayList<Action> Actions = InstanceTable.getTable().getAction(this) ;
               for (Action act : Actions) {
                    if ( act instanceof act_name )
                         act.doIt() ;
               }
               Method [ ] M = currentStateclass.getMethods () ;
               try{
                  for ( i = 0 ; i < M.length ; i++ ) {
                                  :
                       Method m = M[ i ] ;
                       String compare = cstm_name + evName ;
                       if ( m.getName ().equals(compare))
                            currentState = (State) m.invoke( currentState ) ;
                  }
        }
}
```

Java標準のリフレクション機能を
用いてメソッドを呼び出す．
fieldから名前を取得する．

構造リフレクションを用いて
このメソッドがイベントの
数分追加される

図 3　originalStateTransition

ことができるので，修正が必要になった際にコードを修正すれば同時に生成系も修正することになる．コードは設計方針に基き，各処理をメソッドにまとめたことで各処理を区別しているので，修正する際に修正が必要な箇所が分かりやすくなり，保守性が高まると考える．しかし，クラスの書換え，インスタンス生成，登録の動作を全ての図式情報に対して繰り返しおこなうので，実行効率が悪くなると考えられる．

6 おわりに

本研究では，E-AoSAS++ に基づくアーキテクチャからリフレクションを用いてプラットフォームコードの設計をおこなった．リフレクションを用いることで保守性が高いコードを設計することができた．今後の課題としてプラットフォームコードを実現することが挙げられる．

参考文献

[1] 太田将吾, “ソフトウェアアーキテクチャからプログラムコードへの自動変換に関する研究,” 2007 年度南山大学修士論文, 2008.

[2] 千葉滋，立堀道昭, “Java バイトコード変換による構造リフレクションの実現,” 情報処理学会論文誌, vol. 42, no. 11, 2001, pp. 2752-2760.